THE UNIVERSITY
of EDINBURGH

This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.
A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.
This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.
The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.
When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Generative Neural Data Synthesis for Autonomous Systems

Marija Jegorova

Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

This dissertation is submitted for the degree of
Doctor of Philosophy

November 2020

To my loving parents,
Oleg and Viktorija

Acknowledgements

First of all, I would like to thank my academic advisor Timothy Hospedales for his truly supernatural patience, guidance, and support that made completing this degree possible.

I am also very grateful to my dear friends and colleagues at the School of Informatics, especially the lovely people of Machine Intelligence Group, and the group's mascot Lucy Hao Liu for listening to my banter and whining throughout the years and not fleeing the country.

Special thanks go to the members of the tango family, who never failed to share their delightful company, all the joys and sorrows, and a lot of fine meals and wines with me. And finally, I would like to thank the StackOverflow community and Google Search Engine for the technical support.

I am very grateful to everybody involved for their kind friendship and support, and not giving up on me on the way.

Abstract

A significant number of Machine Learning methods for automation currently rely on data-hungry training techniques. The lack of accessible training data often represents an insurmountable obstacle, especially in the fields of robotics and automation, where acquiring new data can be far from trivial. Additional data acquisition is not only often expensive and time-consuming, but occasionally is not even an option. Furthermore, the real world applications sometimes have commercial sensitivity issues associated with the distribution of the raw data.

This doctoral thesis explores bypassing the aforementioned difficulties by synthesising new realistic and diverse datasets using the Generative Adversarial Network (GAN). The success of this approach is demonstrated empirically through solving a variety of case-specific data-hungry problems, via application of novel GAN-based techniques and architectures.

Specifically, it starts with exploring the use of GANs for the realistic simulation of the extremely high-dimensional underwater acoustic imagery for the purpose of training both teleoperators and autonomous target recognition systems. We have developed a method capable of generating realistic sonar data of any chosen dimension by image-translation GANs with Markov principle.

Following this, we apply GAN-based models to robot behavioural repertoire generation, that enables a robot manipulator to successfully overcome unforeseen impedances, such as unknown sets of obstacles and random broken joints scenarios.

Finally, we consider dynamical system identification for articulated robot arms. We show how using diversity-driven GAN models to generate exploratory trajectories can allow dynamic parameters to be identified more efficiently and accurately than with conventional optimisation approaches.

Together, these results show that GANs have the potential to benefit a variety of robotics learning problems where training data is currently a bottleneck.

Table of contents

Nomenclature

Roman Symbols

D Discriminator

G Generator

Acronyms / Abbreviations

ATR Autonomous Target Recognition

BN Bayesian Networks

cGANs Conditional Generative Adversarial Networks [1]

DA Data Augmentation

DCGANs Deep Convolutional Generative Adversarial Networks [2]

FID Freschét Inception Distance [3]

GANs Generative Adversarial Networks [4]

GPNs Generative Adversarial Policy Networks [5]

KDE Kernel Density Estimation

mAP Mean Average Precision

MC-pix2pix Markov-Conditional pix2pix [6]

ML Machine Learning

R2D2-GANs Twice recurrent GANs with double discriminator [7]

RL Reinforcement Learning

RMSE Regularized Mean Squared Error

SIDE-GANs System IDEntification GANs [8]

Chapter 1

Introduction

1.1 Background and Motivation

Within the past few decades, Machine Learning has become increasingly important in computational sciences. This has been enabled by steady advancement of algorithms, parallel and GPU computing capabilities, and especially large datasets for training deep neural networks. Nevertheless, most major successes of ML, such as [9–11] rely on huge training datasets to obtain good results. This data-hunger of existing algorithms is one of their main limitations [12, 13].

This limitation is less of a barrier in domains with access to vast datasets (for instance customer databases of Facebook and Google), but can prove fatal for many real-world applications of Machine Learning, such as drug discovery [14], medical image analysis [15, 16], and robotics [17]. Often there is only a small amount of data available and acquiring more is either impossible or represents a significant difficulty.

The reasons are numerous, however the common factors affecting data availability are the following:

- Commercial sensitivity: if the data needed provides a competitive edge to the owner (i.e., is commercially or national security sensitive), then the owner might not be willing to share it publicly or sell it to the model operator [18].
- Privacy issues arise if the collector and the subject of the data are not the same, because in such case the data owner might not be willing to sell the data to the model owner, especially in cases when the data involves personal sensitive information, such as medical records for instance [19, 20].
- Data acquisition can also be expensive and time-consuming, especially if it involves the use of rare and expensive equipment, or gathering information from real people, or data labels from specialists [21].

Issues particularly relevant to the field of robotics include:

- Data gathering is naturally very time-consuming, since it requires a physical robot to spend time performing some actions, and physical robot operations cannot be easily parallelized or crowd-sourced due to the limited hardware. For instance, in [22] the authors spent 2 months with 6 to 14 robotics actuators to generate 800,000 grasping attempts for one of their experiments. In [23] the authors trained their traffic manoeuvre predictor using data from 1180 miles of driving with 10 different drivers. [24] used 700 robot hours, comprising 50,000 grasp attempts, for training a self-supervised grasping model.

- Additionally, data gathering can require labour intensive experiments or operating the environment might require manually intensive maintenance. From learning to infer liquid properties from robot pouring [25], where each roll-out required an operator to clean-up and refill a cup, to learning by demonstration, where a robot has to be literally taken by the hand to initially perform tasks as a part of the learning process [26, 27].

- Depending on the specific behavioural data required, the data collection can often be very financially costly because of the robot's hardware wear and tear, and potential damage. This can trigger the requirements of repair costs and replacement parts, translating into time and money. Furthermore, gathering data in a real world environment can lead to an even greater chance of damage to, or loss of equipment [28, 29].

1.1.1 Approaches to Data Shortage

What if the additional data collection cannot be conducted? There are a number of ways in which data shortages can be tackled, and the solution is usually case-specific. One way is to keep the training data as they are and attempt to solve the problem at the level of the learning algorithm.

The easiest solution then is using *simpler models*, that are more fit for the small datasets at hand. The lower the complexity of the model, the less likely it is to overfit with a smaller training dataset.

Alternatively, one can use techniques like transfer learning [30, 31] and few-shot learning [32–34], that assume some more data are available in a similar enough source domain(s)[1]. *Transfer Learning* transfers the knowledge learnt in a source domain to

[1]The domain D consists of: a feature space χ and a marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \chi$, [35]. Source domain is used for training a model that should ideally perform well in a (different) target domain.

a target domain (where only a small amount of data is available) to improve learning speed, efficiency, and performance. *Few-Shot Learning* deals with scenarios where the source domain(s) data are labelled and there is an extremely small amount of labeled examples (usually ≤ 5) available in the target domain.

Another way is to treat the data shortage problem is by adding more data into the training set. Since directly collecting more data is ruled out, a variety of methods aim to artificially synthesize additional data:

Sim2Real. In cases where the real data is scarce, but some kind of simulation is available, it could be possible to train in simulation in a way that the resulting model would be able to generalise to a real world environment [36]. This is not always a trivial task because of the reality gap [37]—simulations are never fully capable of capturing the real world. Classic domain randomisation produces a wide distribution over domains in order to train a robust model, applicable in a real world environment. Domain randomisation has been effective in both vision and reinforcement learning [38, 39]. Such an approach allows for generation of any required amount of data, however it raises an issue of tuning the simulation distribution.

Data Augmentation. Conventional data augmentation expands the training data set by using domain-specific knowledge to define class-preserving perturbations on the data, such as rotate/scale in images [40], or different acoustic environments and sound deformations in audio [41]. Stochastic regularisation methods such as dropout augment data at the feature-level by injecting stochasticity. More recent approaches use reinforcement learning to learn the best transformation, and train neural networks to augment data end-to-end [42, 43]. Data Augmentation for imbalanced data can also be done through sampling, such as the Synthetic Minority Over-sampling Technique [44] or Adaptive Synthetic Sampling Approach [45]. Finally, the recent mixup line of work does augmentation by interpolating between both inputs and labels to generate new examples [46].

1.1.2 Data Augmentation with GANs

In this thesis we expand on the GAN-based approaches to data augmentation, and in particular their application to diverse problems in robotics.

The idea behind GAN-based approaches to data augmentation is to fit a generative model to the real training data set, to learn the underlying distribution of it, and then draw samples from it to expand the dataset. Other generative models such as Kernel Density Estimation [47], Variational Autoencoders [48], etc can also be used for that but recent trends show GANs generating higher quality samples [49–52].

In an ideal scenario GANs learn to sample the underlying distribution of the provided training dataset, thus providing an endless source of new diverse data, indistinguishable from real data samples. Data augmentation with GANs has been very successful for image applications [53, 54].

As a rule, GANs operate under the assumption that a sufficient amount and quality of training data is available, meaning the available data are representative of the real data distribution, so GANs can generalise from it. The failure to meet this requirement will most likely end up in lower quality outputs, such as blurriness and/or mode collapse[2].

GANs are conventionally applied to image data for the most part, whether it is generation, completion, or translation [55–57]. The applications to non-image data are still severely understudied, especially for robotics and automation problems. In this work we intend to study this issue and show how this class of methods can be just as useful to generate other types of data. To support this claim we will provide examples of successful application cases, such as generation of sonar scans, and robot control trajectories—both for expanding the behavioural repertoires and for facilitating the dynamic system identification.

Deviating from typical photo-image applications poses some new challenges, specific to the nature of the data of interest. For example, despite some superficial similarities, there is a substantial distinction between typical optical images and the data gathered by sonar sensors:

1. Sonar data are extremely high-dimensional by nature (at least approximately $2 \times 512 \times 300,000$ per mission), it is magnitudes larger than any of the existing super-resolution image GANs have ever been built for.

2. Most of the training data for sonar GANs are commercially sensitive, hence the models have to be able to train on whatever hardware is currently available on site, and cannot rely on the off-site cloud GPUs like some of the image GANs do.[58]

3. Colour consistency is not an issue for sonar data, since they are practically monochrome, however texture consistency within classes is crucial for downstream applications.

In a similar manner there a number of differences between the image data and robot control trajectories. The sensitivity to individual values within a generated sample increases, since these are not sensory but rather control instances in this case.

[2]Mode collapse is a phenomenon where the trained generator only outputting one or very few specific realistic examples instead of a diverse scope of these. Please refer to subsection 2.1.5 for more details.

1. Opposite to the previous set of distinctions in the case of control trajectories the dimension is lower than for classic image data, however:

2. there are new constraints on the validity of the output manifold. For example, one cannot generate trajectories that comprise collisions and self-collisions, bridge joint limits, or otherwise damage a robot.

3. Most importantly for control: despite a the lower dimensionality of the generated data, the constraints on each value within a data sample are much stricter, as any minor noise (that could be imperceptible in case of image data) can render the trajectory parameters useless. For instance, in case of the targeted throwing - noise could render trajectories into not smooth or precise enough, in case of the system identification trajectories - noise could temper with their ability to bring out the necessary inertial parameters of the system.

Besides the obvious benefit of reducing the size of the data required to train sample-inefficient learning algorithms, the work in this thesis explores the following robot and autonomous-system applications of GAN-based data generation:

- Controlled visual simulation: the main purpose is to provide generated data of a specified configuration for training of human operators, by synthesising some specific visual examples of interest.

- Such visual simulation mechanism can also be used for training of autonomous target detection and recognition algorithms.

- In the context of the robotic control, GANs could potentially be used to generate valid diverse behaviours (robot control trajectory outputs, rather than images-like outputs). We can use these to drive exploration processes which are necessary for applications like dynamic system identification.

- Generating bigger behavioural repertoires. Having more than one way to execute a task enables a robot to be robust to obstacles, joint failures, etc. Control-generator GANs can generate diverse ways of performing a task allowing more robust robot behaviours.

1.2 The Research Questions

Extending GAN-based data augmentation to robotics and automated systems in large part will require satisfying the new constraints, posed by the specifics of the application domains. These can be summarised into the following research questions:

Obeying general data quality constraints

Can GANs be adapted to generate realistic looking sonar data of sufficient quality for training ATR models and human operators?

Can GANs be adapted to produce valid robot control trajectories? I.e., without bridging the joint position and velocity constrains, and without self-collisions.

Satisfying task-specific requirements

Can GANs be adapted to generate sonar data of user-specified seabed configuration? How can we achieve the sufficient size and continuity of the images to use them as a simulated output of a full-length underwater missions?

In the context of robot control trajectories, is it possible to produce somewhat successful trajectories for targeted throwing?

In the context of system identification, how can we implement a higher level requirement for the trajectories to be "exciting" enough to bring out the system inertial parameters?

To conclude, in this work we explore the benefits of the non-conventional applications of Generative Adversarial Network class of models, specifically targeting some of the settings where more (or more diverse, or more specific kind of) data could be used.

As a part of this research, we offer a variety of novel GAN methods, such as Generative Policy Networks [5] in Chapter 4, Markov-Conditional continuous image translation with MC-pix2pix [6] and unlimited resolution image generation with R2D2-GANs [7] in Chapter 3, and SIDE-GANs [8], specifically designed for dynamic system identification purposes, in Chapter 5.

1.3 The Structure of This Thesis

This thesis consists of the following 5 chapters:

Chapter 2 gives a top-level overview on GANs providing a brief look at the basic theory and the most prominent variations within this class of methods. We introduce the relevant literature for each problem in the beginning of each chapter.

Chapter 3 focuses on the recursive synthesis of the super-high-dimensional sonar data for providing training examples for human operators, as well as to help train autonomous target recognition systems for object detection underwater. Thus providing an example of automating detection/recognition via using GANs for data augmentation, as well as building a non-commercially sensitive visual simulator.

Chapter 4 explores applying GANs to generation of behavioural repertoires for robot control, potentially useful for domain adaptation. The demonstrated applications are capable to overcome impedences, such as obstacle occluded environments and random damaged joints.

Chapter 5 focuses on generating valid and diverse control trajectories, suitable for improving dynamic system identification (empirically proved at the level of the torque prediction). This results in GAN-generated data improving the exploration of system parameters.

Chapter 6, the conclusion, describes the significance of this work and contains a small discussion on limitations of the proposed approaches, as well as some directions for further research.

Chapter 2

Related Work

This chapter aims at providing a general overview of the underlying class of methods, along with the general literature review, typical problems and applications. Each technical chapter features more specialised literature reviews, corresponding to the specific problems addressed in each of the chapters.

Generative Adversarial Networks (GANs) were originally introduced by Ian Goodfellow in 2014 to address the challenge of learning a neural network-based generative model for complex high-dimensional data [4]. Since then, they have grown into a highly diverse class of methods and have become the most popular way of generating realistic images and video [59]. They have also been applied to image completion [57], super-resolution [55], and style transfer [60, 56, 58, 61]. There have been a number of lesser known applications, such as generation of socially acceptable trajectories of human motion patterns [62] and control policy generation [63, 64, 5]. Nevertheless, the visual data still stays the primary domain of the GAN class of methods.

2.1 Generative Adversarial Networks at a Glance

2.1.1 Vanilla Generative Adversarial Networks

GANs are a powerful yet relatively simple unsupervised learning method for training generative neural models for complex data. The key idea behind GANs is to enable two network system (consisting of the generator and discriminator networks) train in an adversarial fashion, improving via competition. Please refer to the Figure 2.1 for a rough visual explanation.[1]

The generator's task is to produce realistic and diverse examples drawn from the same distribution as the training data set. Specifically it receives a noise vector as an input and

[1] Image credit: `https://github.com/hwalsuklee/tensorflow-generative-model-collections`

produces a data instance (typically an image) as output. We will sometimes call these data "fake" or "synthetic" in future. The generator is trained to maximize the probability of producing data indistinguishable from the real data points in the training set, as estimated by the discriminator. However, the generator is never explicitly exposed to the real training data.

The discriminator receives both the real training data and the generator output. It has to label both as "real" or "fake" (0 or 1). The discriminator is trained to maximize the probability of differentiating real and fake data samples, real provided by the training dataset, and fake by the generator.

The training objective of the discriminator are based on correctly identifying real instances and fake instances. The loss of the generator is based on its ability to fool the discriminator into labelling fake data as real. In other words, the generator G and the discriminator D are playing a two-player minimax game with the value function $V(G,D)$:

$$\min_{G_\theta}\max_{D_\phi} V(G_\theta,D_\phi) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[1-\log D_\phi(G_\theta(\mathbf{z}))] \qquad (1)$$

where $\mathbf{x}$ is the real training data, and $\mathbf{z}$ is the noise, and $p_z(\mathbf{z})$ typically follows a uniform distribution or a multivariate normal distribution with a diagonal covariance matrix, and θ and ϕ are the network parameters for the generator function G and the discriminator D correspondingly.

The training process is iterative. Both networks train from scratch, and in the beginning their outputs are completely random. However, the training is also competitive, so eventually the discriminator learns to classify data better, and generator comes up with more and more realistic outputs. Usually these two networks improve simultaneously, and eventually losses of both networks converge to some local optimas.

From the perspective of Game Theory, the generator and the discriminator are playing a non-cooperative game, where each player tries to minimize its own loss function, which would imply the higher loss for the opponent. So the ideal mathematical solution is the Nash equilibrium, i.e. a point where both losses are optimal with respect to the parameters. Unfortunately, there is no existing algorithms for finding Nash equilibrium for this special case, which features non-convex cost functions and continuous high-dimensional parameters. Hence GANs are typically trained to find a locally optimal solution by applying stochastic gradient descent on each player's cost iteratively [65].

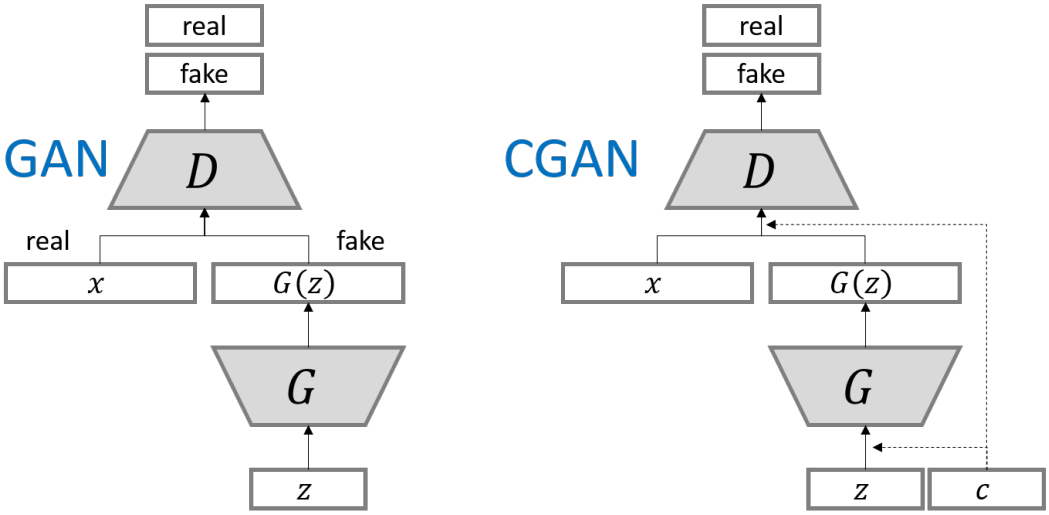

Fig. 2.1 **GANs architecture**: The generator G receives a noise vector z as an input and produces a synthetic data instance $G(z)$ as an output. The discriminator D receives both synthetic data $G(z)$ and real data x and attempts to predict whether it has been presented with a real or fake instance. **The architecture of cGANs** is very close to classical GANs except for the the condition data c that is provided to both the generator and discriminator.[1]

Typically, the output of such iterative training is a well-trained generator that is able to sample the underlying distribution of the training data. The idea is that *at test time* it will generate novel, diverse, and realistic looking data, when executed with a new noise vector input.

2.1.2 Conditional GANs

The original proposal for GANs was to draw unconditional samples form the data distribution. Subsequent work extended GAN to the conditional setting [1], which is important because we often want to draw conditional samples (e.g., sample an image conditional on what type of object should be present, or a robot movement conditional on the goal of the movement). In this thesis we are going to use conditional methods extensively. This is justified by the goal of target-specific real-world applications, that would assume conditional probability distribution.

The idea of Conditional GANs (cGANs) is very straight-forward: now in addition to the original inputs (the noise vectors for the generator and the real and fake data for the discriminator) both networks also receive some sort of condition **c**. The modification to the objective function that **c** causes is minor, and can be expressed in the following way:

$$\min_{G_\theta} \max_{D_\phi} V(G_\theta, D_\phi) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[logD_\phi(\mathbf{x}|\mathbf{c})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[1 - logD_\phi(G_\theta(\mathbf{z}|\mathbf{c}))] \quad (2)$$

There are a few different approaches to feeding the condition into the networks. The earliest one, [1], treated conditions just like a second input and used a simple network architecture like on the Figure 2.1 right for training both - the generator and the discriminator. An alternative to this, [66], is to allow the discriminator to see the condition closer to the output layer, rather than feeding it in as just another input at the bottom layer. Intuitively it makes sense, as the discriminator then considers the condition at a higher level of abstraction. In practice, it does not make much difference at which point the conditioned data is supplied to the network.

2.1.3 DCGANs: Deep Convolutional Generative Adversarial Nets

Deep Convolutional Generative Adversarial Networks (DCGANs) is a popular design for a GAN architecture, it was proposed to expand on the internal complexity of the generator and the discriminator. DCGANs extended the success of the Convolutional Neural Nets in computer vision applications to unsupervised learning [2].

They introduced the following modifications to classic vanilla GANs:

- For the discriminator replace any pooling layers with strided convolutions, for the generator - with fractional-strided convolutions. These are supposed to allow the network to learn its own spatial downsampling (for discriminator) and upsampling (for generator) [67, 2].

- Use batchnorm in both the generator and the discriminator. It helps to deal with training problems that arise due to poor initialization and helps gradient flow in deeper models [68, 2].

- Remove fully connected hidden layers for deeper architectures. Authors of DCGANs paper found global average pooling increased model stability but hurt convergence speed. A middle ground of directly connecting the highest convolutional features to the input and output respectively of the generator and discriminator worked well. The first layer of the DCGAN generator, which takes a uniform noise distribution as input, could be called fully connected as it is just a matrix multiplication, but the result is reshaped into a 4-dimensional tensor

and used as the start of the convolution stack. For the discriminator, the last convolution layer is flattened and then fed into a single sigmoid output [69, 2].

- Use ReLU activation in generator for all layers except for the output, which uses Tanh, and LeakyReLU activation in the discriminator for all layers [70, 2].

For a default template architecture, as presented by the original DCGAN paper, please refer to Figure 2.2. Standard input is a $100D$ noise vector (uniform random $(-1, 1)$, although the spherical noise also can be used). In this example the output is an RGB picture of size 64×64, or a tensor of size $64 \times 64 \times 3$.

The loss function takes the same form as the vanilla GAN one, provided in Equation (1). The usual optimiser is Adam, with learning rate 0.0002 and momentum term of 0.5. The generator and the discriminator usually mirror each other, both networks feature a single fully connected and a few (in current example four) convolutional layers. The official example is presented in Figure 2.2, with corresponding sizes and stride 2 for convolutional layers, batch size of 64. Training epochs are usually varying depending on the quality of generated samples, with the default number at 25.

Depending on the type of data, occasionally discriminator trains much faster than generator suppressing generator training as a result, since they are interconnected through the loss function. In this case either discriminator size should be reduced (smaller number of layers usually) or generator should be allowed more training updates compared to discriminator.

2.1.4 GANs for Image Translation: Cycle-GANs and pix2pix.

Image-to-Image translation is one of the popular applications of the generative adversarial nets. It uses conditional GAN setup to change image in domain X and translate it into domain Y. Depending on the training data available it can be either unpaired translation (for unpaired sets of images, learning to translate in an unsupervised fashion) or paired translation (for images paired across the style sets, traning in supervised fashion). Unpaired image translation is usually done with Cycle GANs [60], and paired translation is usually done with pix2pix [56].

Cycle GANs are used in case when the training images are of unpaired across domains. They feature four networks - two generators and two corresponding discriminators. Generator G_θ translates images from domain X into domain Y, second generator F_γ translates images from domain Y into domain X. The discriminators try to distinguish between real and fake images for their corresponding generators: $D_{\psi Y}$ tries to distinguish y from $G_\theta(x)$ and $D_{\phi X}$ tries to distinguish x from $F_\gamma(y)$. Please refer to

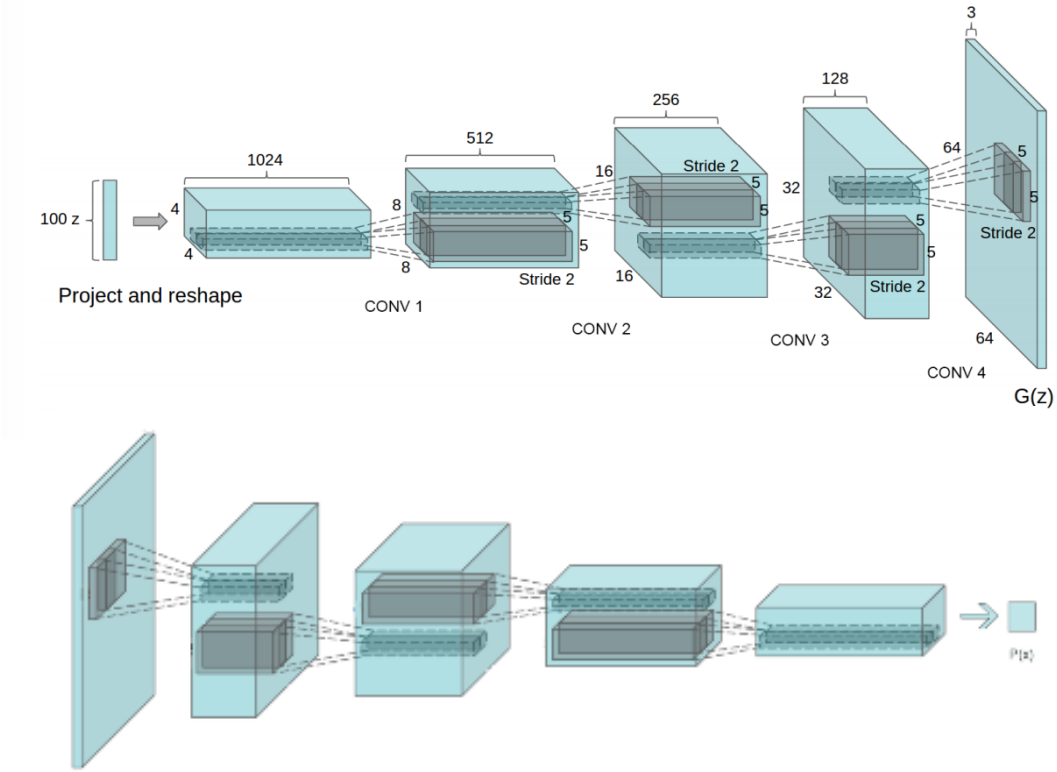

Fig. 2.2 **Template DCGANs architecture. Top:** the generator G receives a noise vector z as an input, takes it through a fully connected layer & reshaping, and then four (de)convolutional layers with ReLU activation functions and Tanh activation for the last layer. G produces a synthetic data instance $G(z)$ as an output. **Bottom:** the discriminator D is in essence a mirror reflection of the generator. It receives both - synthetic data $G(z)$ and real data x, pulls them through four layers of convolutions, concluded by a fully connected layer. D is meant to predict whether it has been presented with a real or fake data instance. [2]

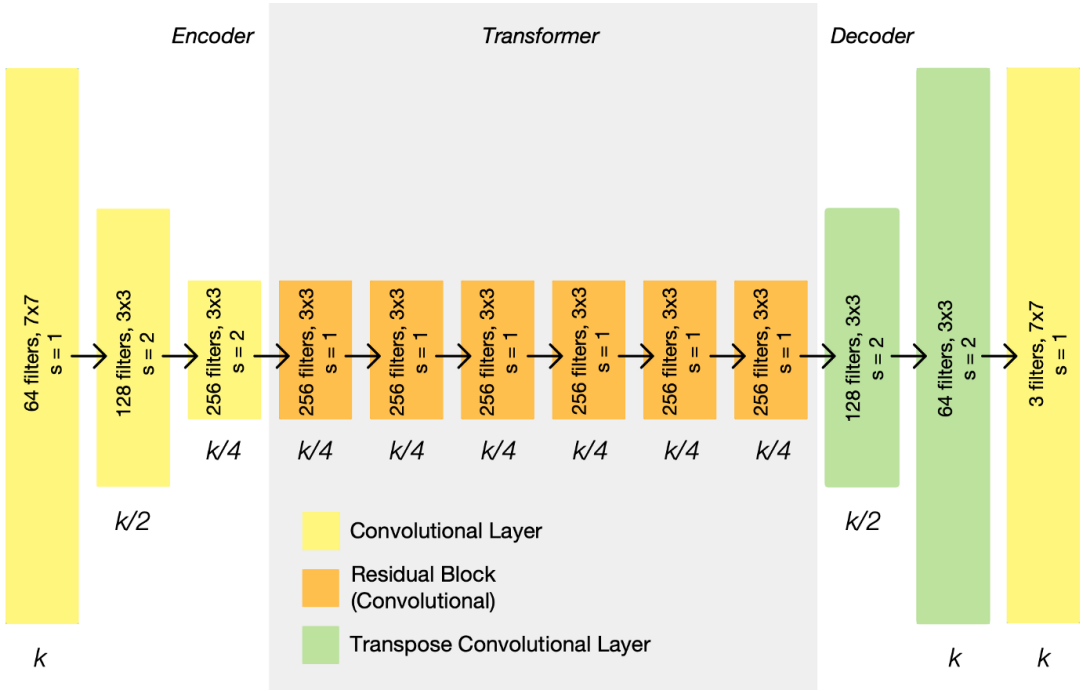

Fig. 2.3 **Example of CycleGAN architecture: Generator.** The representation size that each layer outputs is listed below it, in terms of the input image size, k. On each layer is listed the number of filters, the size of those filters, and the stride. Each layer is followed by an instance normalization and ReLU activation.[2]

the example architectures of CycleGAN generators and discriminators in Figures 2.3 and 2.4. [2]

The objective function consists of adversarial loss (typical GAN loss function) and cycle loss (specific to cycle GANs), computed as following:

$$
\begin{aligned}
L(G_\theta, F_\gamma, D_{\phi X}, D_{\psi Y}) &= L_{GAN}(G_\theta, D_{\psi Y}, X, Y) + L_{GAN}(F_\gamma, D_{\phi X}, Y, X) + \lambda L_{cyc}(G_\theta, F_\gamma) \\
&= \mathbb{E}_{y \sim p_{data}(y)}[\log D_{\psi Y}(y)] + \mathbb{E}_{x \sim p_{data}(x)}[1 - \log D_{\psi Y}(G_\theta(x))] \\
&\quad + \mathbb{E}_{x \sim p_{data}(x)}[\log D_{\phi X}(x)] + \mathbb{E}_{y \sim p_{data}(y)}[1 - \log D_{\phi X}(F_\gamma(y))] \\
&\quad + \lambda \left(\mathbb{E}_{x \sim p_{data}(x)}[\|F_\gamma(G_\theta(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G_\theta(F_\gamma(y)) - y\|_1] \right)
\end{aligned}
\tag{3}
$$

Authors of the original paper show that both adversarial loss L_{GAN} and cycle consistency loss L_{cyc} are crucial for quality of the results, moreover, single cycle is not enough to regularise training for such an under-constrained problem.

Training is done with Adam optimiser, learning rate 0.0002, and the batch size in the original paper was set to 1 (that would largely depend on the training data). Once

[2] Figures credits:
https://towardsdatascience.com/cyclegan-learning-to-translate-images-without-paired-training-data-5b4e93862c8d

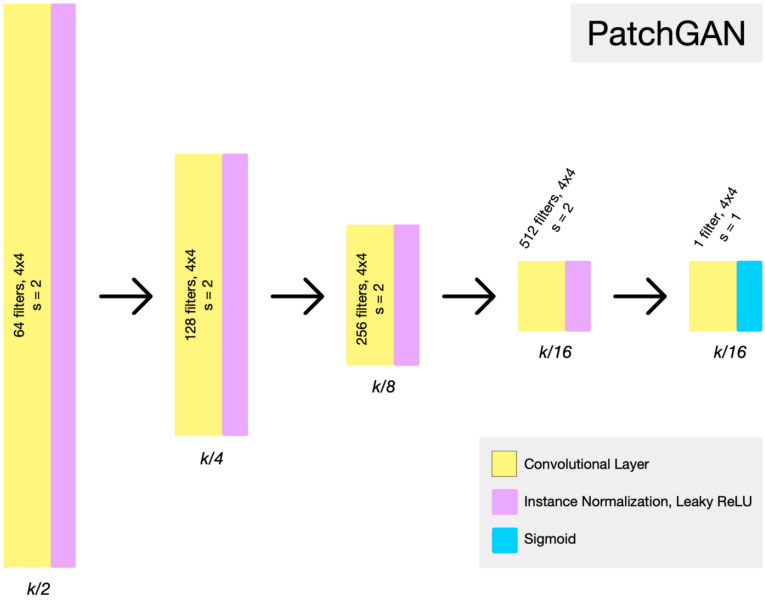

Fig. 2.4 **Example of CycleGAN architecture: Discriminator - PatchGAN.** It is a fully convolutional network, that takes in an image, and produces a matrix of probabilities, each referring to the probability of the corresponding "patch" of the image being "real" (as opposed to generated). The representation size that each layer outputs is listed below it, in terms of the input image size, k. On each layer is listed the number of filters, the size of those filters, and the stride.[2]

the training is complete, both discriminators get discarded and both trained generators can be used for translation in corresponding directions. G for translating images from domain X into domain Y, and F mapping domains $Y \rightarrow X$.

pix2pix is the image-to-image architecture for the cases when paired training data are available. In this case the architecture is more compact, featuring only two networks (generator and discriminator), since the problem is better constrained. It is still a conditional GAN setup, the main modifications are down to the generator architecture - it uses either an encoder-decoder combo or a U-Net (which is nearly identical to encoder-decoder), but with the skip connections between each layer i in encoder and layer $n-i$ in decoder.

Original pix2pix paper provides a few examples of architectures, here we present one of them, please refer to Figures 2.5 and 2.6 for more details: [3]

Generator:

encoder: C64-C128-C256-C512-C512-C512-C512-C512

decoder: CD512-CD512-CD512-C512-C256-C128-C64

Discriminator:

C64-C128-C256-C512

[3]Figure credits: `https://neurohive.io/en/popular-networks/pix2pix-image-to-image-translation/`

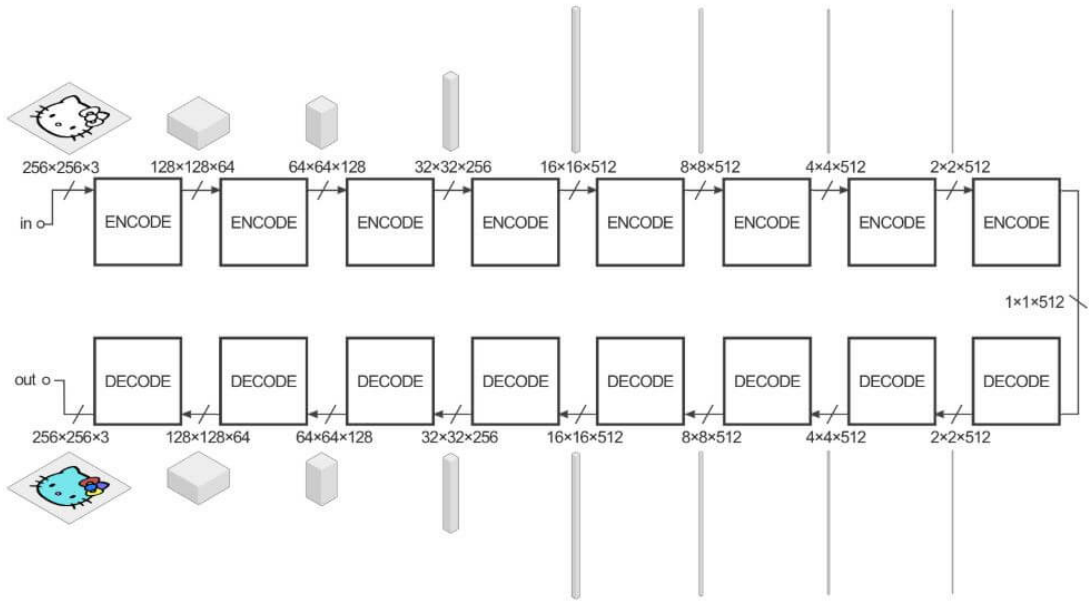

Fig. 2.5 **Example of pix2pix encoder-decoder generator:** it is a conditional architecture, where input / condition is the raw image (from *A* domain) to be translated (into *B* domain) and the output is the translated version.
Batch-Normalization is not applied to the first *C*64 layer in the encoder. A convolution is applied after the last layer in the decoder to map the number of output channels, followed by a *Tanh* function. All ReLUs in the encoder is leaky, with slope 0.2, while ReLUs in the decoder is not leaky. The U-Net architecture is identical except with skip connections between each layer i in the encoder and layer $n-i$ in the decoder, where n is the total number of layers. The skip connections concatenate activations from layer i to layer $n-i$.[3]

Discriminator is a PatchGAN, just like in case of the CycleGAN. Discriminators always follow the same basic architecture, with depth varied to modify the receptive field size.

Training follows the typical GANs procedure - both networks train from scratch in an adversarial fashion, both networks weights are usually initialised from the Gaussian distribution $N(0,0.2)$. The default optimiser is usually Adam with learning rate 0.0002 and $\beta_1 = 0.5$.

The loss function is similar to that of the typical conditional GAN, where the condition is expressed via the raw image of domain X to be translated, and a translated image $y \in Y$ is the final output of the trained model, being assessed by the discriminator for realism:

$$L_{cGAN}(G_\theta, D_\phi) = \mathbb{E}_{x,y}[\log D_\phi(x,y)] + \mathbb{E}_{x,y}[1 - \log D_\phi(G_\theta(x,y))] \tag{4}$$

The trained model will only use the trained generator network and facilitate the image-to-image translation from domain X into domain Y.

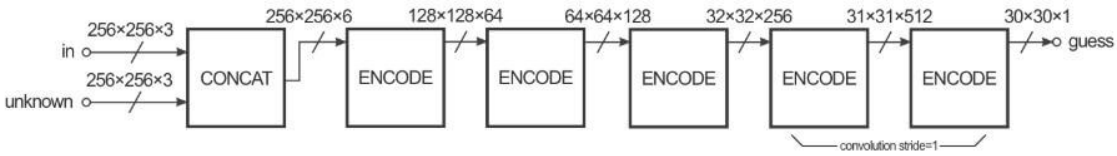

Fig. 2.6 **Example of pix2pix discriminator architecture:** Discriminator takes an input of a raw image from the domain X and an unknown translated image from the domain Y coming either from the generator or from the real training data set.
A convolution is applied after the last layer to map to a 1-dimensional output, followed by a *Sigmoid* function. BatchNorm is not applied to the first $C64$ layer. All ReLUs are leaky, with slope 0.2.[3]

CycleGANs are generally inferior to pix2pix in the translation quality (please refer to the Chapter 3 for some of the visual comparisons), and pix2pix is a preferable image translation method if only the paired training data set is available.

2.1.5 Potential Issues During Training

There are quite a few issues and limitations that often arise during GAN training, the main of which are as follows:

Data. As most neural network-based models, GANs are very much data dependent. Not only do they need enough training data, but often they also need specifically diverse training data in order to achieve good results.

Mode Collapse. Sometimes the training converges to a bad local optima where the generator only outputs one or very few specific realistic examples, regardless of the noise input it receives. It is a relatively rare failure mode to encounter nowadays with the more recently proposed DCGAN modifications [2]. However, it was a very common problem back when GANs were first introduced. Most of the modern techniques [2, 55, 58, 49] are built with convolutional layers, and often with batch norm and weight clipping in place to ensure the stability of the training process.

Loss convergence in relation to the sample quality. Unless one is using some specifically designed model, such as Wasserstein GAN [71], the losses of the generator and discriminator are not meant to be particularly representative of the performance of the model. Also, there are no loss convergence guarantees, which in practice means either generator or discriminator can grow too strong too quickly, and block the other one from training properly. Usually however, depending on the data, initialisation, and hyper-parameters of the network, the losses of the generator and the discriminator will converge to some stable values. These values themselves are not informative, and not important. Such convergence signifies that the GAN has found some optima, and cannot

improve further. There is still a chance that this is a bad local optima, so the results should be checked for a *mode collapse*, but otherwise the training is complete.

Performance Evaluation. Evaluating generative models is hard in general since there is no single correct output. Conventional way of evaluating generative models is the average log-likelihood on a held-out validation set. However for some models log-likelihood is hard to compute or even approximate [72]. For instance, latent-variable models might involve solving complex integrals to compute the likelihood. These models may still be trained with respect to a different objective that is related to log-likelihood, such as lower bounds on the log-likelihood [73], noise-contrastive estimation [74], probability flow [75], maximum mean discrepancy (MMD) [76, 77], or in our case approximations to the Jensen-Shannon divergence (JSD) [4]. The alternatives are *assessing the generated samples* in various ways, interpreting model parameters, and *evaluation of model performance on surrogate tasks* [72].

Since training GANs does not directly involve log-likelihood, they cannot be evaluated in this way. Hence we have to stick to directly evaluating the generated data, or assessing their performance with applications to other tasks.

The first approach includes many types of assessment, including visual, inception score [65], and Freschét Inception Distance (FID) [3].[4] Inception Score (IS) was the initial (and now outdated) way of numerically assessing images generated by GANs. I was proposed by [65] to apply an Inception-v3 network, [78], pre-trained on ImageNet, [79], to generated images and then comparing the conditional label distribution with the marginal label distribution, like so:

$$IS = exp(\mathbb{E}_x \sim p_g D_{KL}(p(y|x) \| p(y)) \tag{5}$$

Higher inception scores correspond to a larger KL-divergence between the two distributions, which is better.

The FID is comparing the statistics of generated samples directly to the real samples in the following way:

$$FID = \|\mu_r - \mu_g\|^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{(1/2)}) \tag{6}$$

[4] Freschét Inception Distance (FID) [3] is a heuristic for measuring the difference between the real and synthetic image distributions. However, for the FID number to be of at statistical significance the available data-sets should be several thousands of images in size for both real and synthetic sets. This is not always an option in real life, where up to a thousand (if not a few hundreds) of training images is all there is for real data.

where $X_r \sim \mathcal{N}(\mu_r, \Sigma_r)$ and $X_g \sim \mathcal{N}(\mu_g, \Sigma_g)$ are the 2048-dimensional activations of the Inception-v3 pool3 layer for real and generated samples respectively.[5] Lower FID is better, corresponding to more similar real and generated samples as measured by the distance between their activation distributions.

This is usually more relevant for immediately visualisable data, which we have in Chapter 3, where we provide human assessment scores[6] and FID for the generated images and baselines.

Most of our applications involve the further use in by other autonomous systems, so we can use the second option for generative model assessment and evaluate the success of our models via performance gain of the downstream system consuming our data.

2.2 A Thousand and One Variation of GANs

Since the original paper came out in 2014, multiple variations have emerged in order to improve upon various shortcomings [2, 71, 80]. As mentioned before, historically the original GANs were not particularly stable to train.

A huge improvement was brought by turning GANs into Deep Convolutional Networks, i.e. DCGANs [2]. Amongst other things, the modifications to the original GANs included batch normalisation, non-fully-connected hidden layers, and strided or fractional-strided convolutional layers. DCGANs significantly improved the stability of GANs, but some other typical problems of GANs, for instance the risk of the mode collapse and the lack of convergence guarantees were not solved completely.

The next big milestone for GANs was the release of Wasserstein GANs [71]. They have addressed stability of the convergence and interpretability of the loss functions. One of the main features was adding Wasserstein distance to the loss function, which made the loss functions correlated with the image quality and made them more likely to converge, hence the improved stability. As a result, WGANs are generally less dependent on the network architecture and batch normalisation.

Wasserstein GANs were closely followed by the Improved Wasserstein GANs, which suggested some alternative way of clipping weights—they penalise the norm of the discriminator gradient with respect to its input. They claim better image quality based on these improvements [80].

[5] Other layers of the Inception-v3 network can also be used, with different dimensions. At least d samples are requiring for estimation of the Gaussian statistics for d-dimensional features.

[6] Involving people in the data quality assessment can be expensive, time-consuming, and is not always a feasible option for specialised data (e.g., medical imagery), where specialists are needed to properly assess the quality.

Authors of f-GANs [81] took the more general approach, treating GANs as a part of variational divergence estimation approach. They have shown that these can be trained using any f-divegence, for instance Pearson Chi-squared, Kullback-Leibler, reverse Kullback-Leibler, etc.

However, a more recent study shows that assuming the right fine-tuning they all perform roughly the same [82]. Hence, in chapters 4 and 5 we stick with classical Deep Convolutional GANs, being the most conventional stable version [2], as a basis for further modifications.

Aside from the stability modifications that were meant to improve upon the classic GAN, there is currently a considerable number of GAN-based architectures built for specific purposes. In fact, the GAN-Zoo list hosted on GitHub[7] states almost 500 published variants, and this list continues to grow. This chapter will not cover all of them, but to give reader a brief idea of the scope, we will attempt to provide a high-level list of fields that are currently explored or being explored:

Image Applications include classic image generation [83], super-resolution [55], image completion [84], and image-to-image translation for aligned [60] and unaligned [56, 58, 61] datasets (both based on U-shaped nets). Recently, there has been a lot of focus on improving the size and quality of the image generation [49, 85], however the largest of them are still much too small for some of the applications in Chapter 3.

Text Generation [86, 87] with GANs is usually based on architectures built in some sort of recurrent setting because of the sequential nature of the data. Some work has been conducted on text-to-image translation as well [88].

Video Generation and Translation [59, 89] is also a growing field at the moment, for instance generation of image and video based on audio input [90, 91].

Behaviour Generation. Other applications span from human social trajectories [62], to generating single agent policies in an imitation learning framework, and also some attempts on single or a few control policies generation[63, 64], and now our new method for learning control policy distributions [5], proposed in Chapter 4.

2.3 Alternative Generative Models

All of the statistical / machine learning classification models are either generative of discriminative [92]. Discriminative models, given the observation $X = x$ and target variable Y only model the conditional distribution $P(Y|X = x)$. Generative models

[7] GAN-Zoo list URL: `https://github.com/hindupuravinash/the-gan-zoo`

attempt to learn the full joint distribution on $X \times Y$ $P(X,Y)$ [93], which means that they can be used to generate instances of observations and targets (x,y).

Generative models might be the oldest semi-supervised models [94], so there is a considerable number of them. We can loosely place them into two big classes: neural and shallow or non-neural generative models.

2.3.1 Shallow / Non-Neural Generative Models

This class mostly includes conventional probabilistic generative models, aka Bayesian Networks (BN) [95, 96]. They tend to perform poorly on high-dimensional data, which is a serious issue for the purposes of this work. There is a number of special cases relevant for data generation, such as Mixture Models, Hidden Markov Models, etc, that we would like to cover briefly:

- *Mixture Models* (often Gaussian Mixture Models) are probabilistic models that assume there exist sub-populations within the general population, and attempt to model these as a mixture (a weighted sum) of multiple multi-dimensional probability distributions.[8] The mixture distribution can then be sampled for the new data. Mixtures of models are often used in fields like business analytics and market segmentation [100]. However, it is hard to generate high-dimensional natural image data with them.

- *Hidden Markov Models (HMMs)* provide a way of modelling the data that is generated in a sequence [101, 102]. Because of its sequential nature HMMs are often used for generation and analysis of biological, medical, and financial time-series [103–105]. Moderate to large dimension estimations still remain problematic to HMMs, mostly because of the challenges arising from the hidden nature of the states [106].

- Averaged One-Dependence Estimators were developed to address the attribute independence in the Naive Bayes classifier. It often performs better than the latter at a trade-off of the computational complexity, which is $O(nk^2)$, where n is the number of training samples and k^2 is the dimensions of the training data, which makes it too computation-intense for higher dimensional data [107].

- *Latent Dirichlet Allocation* is useful for modeling specifically discrete data. This is because it explains the similarities within sets of observations by them belonging to unobserved discrete groups. The natural drawback of it is the

[8] They were popularised in 1973 by Duda and Hart in [97], although the first explicit reference to the mixture decomposition problem dates back to 1894 [98, 99].

requirement to specify a number of such groups either manually, or through sampling some distribution, which might be not objectively representative of the real distribution [108].

- We would also like to separately mention *Kernel Density Estimation (KDE)*, as we employ it as one of the baselines in Chapter 4, for our lowest-dimensional application, to demonstrate the advantage of our GAN-based generative model. KDE can be thought of as an extreme case of a mixture of Gaussians, with a single Gaussian per point. It is a non-parametic way of estimating probability density function of a random variable [47], a reliable alternative for producing data of smaller dimensionality introduced in 1956. Kernel density estimation is not only computationally intense, but also suffers from the *curse of dimensionality*, i.e., as the dimension increases the number of data points needed to get a reliable density estimator grows exponentially [109]. The other issue is that any small change in the training data can cause large fluctuations in the estimated density, and some regularisation needs to be done, which is usually reflected through the choice of the smoothing parameters, and turns a non-parametric KDE into a parametric model [109].

All of these BN methods, despite having their individual strengths and weaknesses, share the property of not being particularly good at modelling raw unstructured higher-dimensional data, such as images, other sensory data, and robot control policies. Hence not only they are not the best choice for the applications presented in this work, but also, on a larger scale, they got left behind by neural generative models in the area of data augmentation.

2.3.2 Deep Neural Generative Models

This is a class of methods utilising neural networks in their architectures, and hence is a bit more flexible than the generative models presented in the previous subsection. GANs belong to the deep neural generative class of models. The other big families in this class are Deep Belief Networks, Deep Boltzman Machines, Variational Auto-Encoders, and Flow-based models.

Deep Belief Networks (DBNs) are generative graphical models with many hidden layers of latent variables, represented by Restricted Boltzman Machine (RBM), usually having last layer as a classifier [110]. They are widely used in data completion and denoising [111, 112], they deal with small amounts of data generally better than GANs, due to their Bayes nature [113]. The reasons why they are significantly less popular than GANs, especially with application to more complex data modelling, are slower

convergence (because of the Gibbs sampling) and being trained mostly on the Maximum Likelihood principle that can become numerically unstable in practice [113, 71].

Deep Boltzman Machines (DBMs). The key difference between the DBMs and DBNs is that DBNs are directed graphical models, and DBMs are undirected [114]. Hence unlike DBNs, they approach training and inference in both directions. Their speed is their main limitation, because the maximum likelihood in DBMs is intractable, and some approximations need to be made, usually approximating gradients of the log-likelihood using Markov chain Monte Carlo (MCMC) [115, 116].

Variational Autoencoders are a family of directed probabilistic graphical models that belongs to the deep latent variable models group. Because the evaluation and/or optimisation of the likelihood might be intractable, it uses strong assumptions about the latent variables distributions. More specifically introducing variational approximations to the posterior and optimizing a stochastic lower bound to the log-likelihood [48]. In comparison, training GANs does not explicitly involve the likelihood function computations, and hence does not have these problems. However, because of that the performance of VAEs can be evaluated easier than GANs. Although VAEs are usually assumed to be more stable at training, GANs so far have shown higher fidelity of the generated data. [117, 49].

Exact likelihood models (Flow-based models) are applying multiple invertible transformations to a data sample from prior, which means computation of the exact log-likelihood of observations is possible [118–120]. This means better evaluation and more stable training, which results in a comparable visual quality to GANs, however it comes at at a massive trade-off in speed. For example, GLOW [120] took 2 weeks of training on 40 GPUs to generate 256×256 celebrity faces, and the resulting model had about 200 million parameters. In contrast, progressive GANs [85] took 4 days on 8 GPUs, using about 46 million parameters, to generate 1024×1024 images for a similar training set. So Glow needed 17 times more GPU-time, and 4 times more parameters for 16 times smaller output [121].

To briefly summarise this section: GANs are generally more successful with moderate-to-higher dimension data generation than shallow / non-neural generative models, produce higher fidelity output compared to Variational Autoencoders, and train more efficiently than Flow-based methods.

Chapter 3

GANs for High-Dimensional Sonar Image Simulation[1]

Deployment and operation of autonomous underwater vehicles is expensive and time-consuming. High-quality realistic sonar data simulation could be of benefit to multiple applications, including training of human operators for post-mission analysis, as well as tuning and validation of autonomous target detection and recognition (ATR) systems for underwater vehicles. Producing realistic synthetic sonar imagery is a challenging problem as the model has to account for specific artefacts of real acoustic sensors, vehicle attitude, and a variety of environmental factors. We propose two novel architectures for generating realistic-looking sonar side-scans of full-length missions, called Markov Conditional pix2pix (MC-pix2pix) and double-recursive double-discriminator GANs (R2D2-GANs, or "R2D2" for the sake of conciseness). Quantitative assessment results confirm that the quality of the produced data is almost indistinguishable from real. Furthermore, we show that bootstrapping ATR systems with MC-pix2pix/R2D2 data can improve the performance. Synthetic data can be generated at between 35 and 280 times faster than real acquisition speed (depending on the desired resolution of the sonar). The proposed methods also allow for the full user control over the topography of the generated data.

3.1 Introduction

In underwater environments, sonars are often preferred over other sensors due to the high density of organic material and inorganic dust that can restrain optical visibility.

[1]This chapter mostly consists of two parts - MC-pix2pix method published in IEEE International Conference on Robotics and Automation 2020, and R2D2-GAN, which is accepted for publication in IEEE OCEANS 2020 (Singapore).

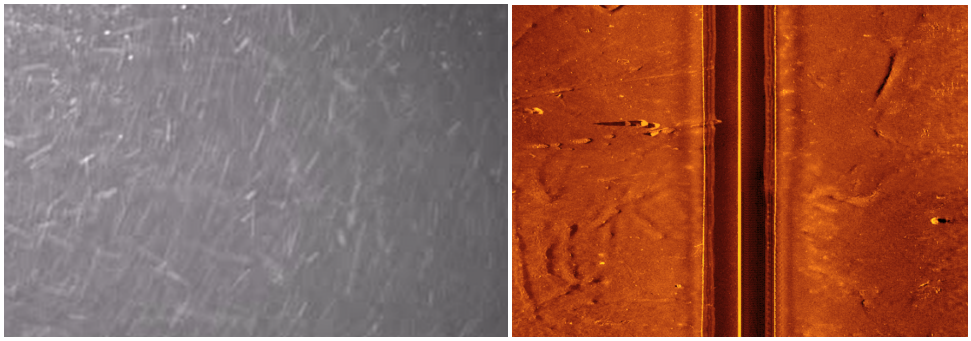

Fig. 3.1 **Examples of underwater sensors. Left:** optical camera - an example of visibility in northern seas, this effect is called *marine snow*, it is caused by high density of fine floats. In cases of poor visibility, sonars are often preferred over the cameras as more perceptually robust. **Right:** sonar side-scan. Port (left) is real, starboard (right) is synthesized.

This effect is call marine snow, an example is provided in Figure 3.1, left. Because of their perceptual robustness, sonar sensor data is heavily relied upon for tasks such as object localization, oil-pipe and infrastructure inspections, search and rescue, and other commercial and military applications. Please refer to Figure 3.1, right, for an example of a small snippet of such data.

A vast amount of data is required to construct detection and recognition models for automating most of these applications. Underwater data collection is expensive, time-consuming, and in most cases commercially sensitive. A means of synthetically creating such data would be highly beneficial to the underwater sensor processing community, as it would mitigate the costly process of data collection by instead making better use of the available real training data.

Existing techniques for image synthesis, such as generative adversarial networks (GANs) [4] have recently grown capable of producing and enhancing images of high resolution (e.g. 2048×1024 by pix2pixHD [58]). However, typical underwater survey missions sonar images usually exceed the image resolution of $300{,}000\times512$ pixels. We propose the Markov Conditional pix2pix (MC-pix2pix) method which, to our knowledge, is the first method capable of generating realistic sensory output for full-length missions, given a small amount of initial training data. Crucially, such generation runs 280 times faster than acquisition on the real hardware, resulting in a realistic and faster than real-time simulator. We then further build upon MC-pix2pix, introducing R2D2-GANs, method which in principle scales to sonar data of any chosen resolution.

To demonstrate the utility of our approach, we provide quantitative results in two extrinsic evaluation tasks: (i) the synthetic data is almost impossible to distinguish from real data for domain experts, thus enabling training of teleoperators without using real

hardware; (ii) significant performance gains are achieved when using this synthetic data to augment training datasets for autonomous target recognition (ATR) in a variety of seabed conditions. The results presented in here are produced with Marine Sonic sonar side-scan data (for the lower resolution examples) and with EdgeTech sonar data (for the higher resolution examples). However the methods themselves are sonar-agnostic.

3.2 Related Work

GANs [4] are a class of neural network models for the realistic data generation. Since their initial introduction in 2014, a large number of extensions have been proposed for various applications, primarily focused on realistic image and video generation [83, 55, 88, 59, 89], where only a limited amount of training data is available. In contrast to these tasks, there is comparatively little work investigating how GANs can be of benefit in robotics. Although robots that use image recognition in domains where training data is scarce may benefit from the conventional applications of GANs. Some applications that are more relevant to robotics include GAN-based approaches to imitation learning [63, 64], which allow robots to efficiently learn a single policy or a discrete set of policies from demonstration, and direct generation of robot control policy repertoires [5], a technique that enables sampling from the continuous target-conditional distributions over the control policies within a scope of a given task.

Facilitation of the user-controlled simulation requires some form of the information transfer from a specification of a desired scene content to the actual image. GANs have been extensively used for style transfer and image-to-image translation, beginning with cycleGANs for transfer between unpaired images [60]. However, on paired image translation problems – the task we are primarily concerned with – pix2pix [56] and its subsequent variations [58, 61] are known to perform considerably better. No current image translation methods can be directly applied to full-mission sonar data because of the extremely high resolution. The size of the full image to be generated is usually in excess of 300,000$\times$512 pixels – roughly the amount of data generated by a short two hour training mission. Our method solves this problem by producing such image in a piece-wise sequential manner, and ensures continuity of the output through the use of a Markov assumption. The use of Markov assumption here is justified by the temporal nature of the real data acquisition during real mission, as well as by the general spatio-temporal continuity of the required data.

The previous attempt of generating sequential data with GANs, recurrent GANs (RGANs) and recurrent conditional GANs (RCGANs) [122], focused on medical sequence generation. This has been accomplished through the use of recurrent neural

network architectures. There are two issues with applying these techniques to the problem of synthetic sonar imagery generation. Firstly, we require the control over the topography. So the model architecture would need to be modified for image translation with convolutional layers, which would further require one to use backpropagation through time for training the network, rendering the training process computationally intractable for the size of data that we work with. Secondly, RGAN and RCGAN are designed to produce semantically realistic sequences, whereas we require perceptually realistic image sequences. Additionally, the nature of the sonar imagery suggests that the Markov assumption alone is enough for the coherency and continuity.

A small number of papers address the underwater robot perception problems with GANs: the work of [123] shows cycleGANs enhancing synthetic target objects for embedding them into the real sonar images in order to train an ATR system, while [124] proposes a method for refining underwater optical video images rather than generating new acoustic imagery.

Until now the applications of GANs to underwater sensory data were mostly enhancing the imagery, either optical or acoustic, rather than generating brand new data. MC-pix2pix is also the first model of its type addressing the generation of a whole mission's worth of data rather than separate smaller images.

3.3 Problem and Motivation

3.3.1 Why generate sonar images?

The key application for high-quality simulation is bootstrapping autonomous target detection and recognition (ATR) methods when training data is scarce, or some types of seabeds are underrepresented in the real training set. In section 3.4.5 we show improvements in the ATR performance when generating a variety of seabeds and introducing them into the ATR training together with the available real data.

Realistic simulation could also benefit the training of teleoperators for mission planning and interpretation of sonar imagery, replacing the costly real data collection.

3.3.2 Synthetic framework for training of the vehicle operators

The simulation pipeline, presented in the Figure 3.2, assumes that the training instructor marks the regions of the map with a specific topography, such as rocks, ripples, clutter, and objects of interest. The trainee operator is presented with this map without the target objects marked, and creates a route over it that should allow the robot to locate these hidden objects. Given semantic maps provided by an instructor and the route

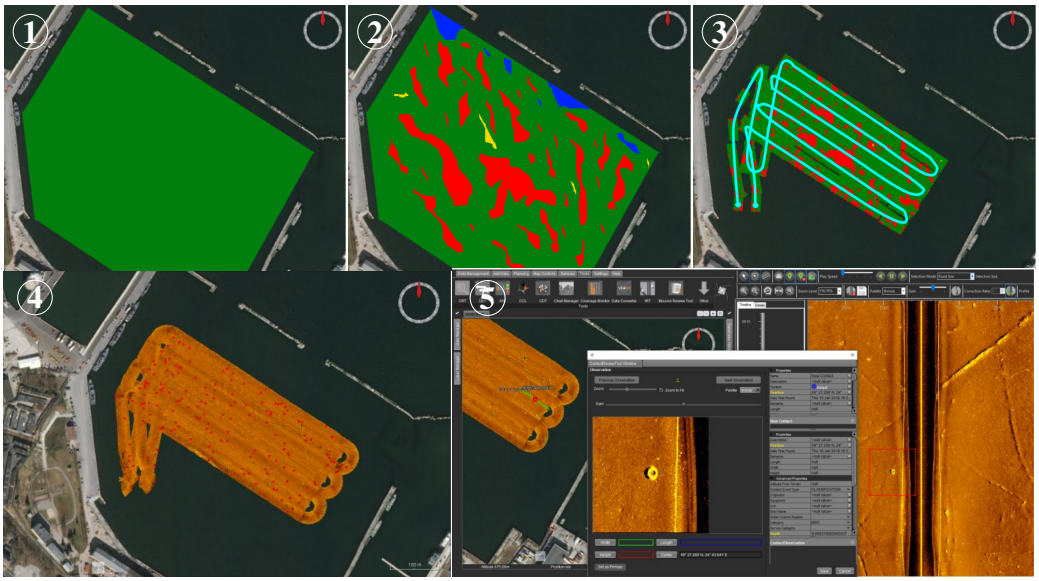

Fig. 3.2 **Pipeline:** 1-2. Training instructor labels the map regions with desired textures and target locations. 3. Trainee operator creates a route across a given map with an objective of collecting data for locating hidden targets. 4. The model receives semantic maps and route as inputs and outputs synthetic sonar data for the entire mission. (In real life vehicle completes the mission delivering the sonar data collected.) 5. Example of sonar images when inspected by a human operator.

created by trainee operator, the purpose of our technique is to generate realistic seabed scans for the entire mission. Methods such as [123] can be employed to embed the target objects in the requested locations in the synthetic seabed scans, and can then be displayed to the trainee operator for visual inspection and object detection, just like during a real post-mission analysis.

The emphasis of this work is on keeping synthetic data maximally consistent and realistic, whilst achieving the highest generation speed possible.

3.3.3 Problem Specification

For the task described above the following requirements should be considered:

- Realistic looking synthetic data generation: the main focus of this work. Our method is based on GANs because they have been identified as the current best approach for generating realistic imagery.
- Spatial coherency: imagery of the entire mission should appear continuous and consistent. The paired nature of pix2pix guarantees consistency within topographical features represented as different labels in semantic maps. Additional

conditions are introduced in section 3.4.3, and improve the continuity of the output further.

- Viewpoints invariance: the same section of the map should appear texture-consistent when observed from different viewpoints.

- Speed of generation: in practice, for faster than real-time simulations, our model should be significantly faster at test time than real-time sonar data acquisition.

3.3.4 Why pix2pix? Why other GANs did not do?

There is a number of GAN-based architectures that we could have chosen for this task. Maybe not out of the box, but to be extended by Markov condition, since in principle it is not bound by a specific choice of the GAN architecture. Here we offer a brief overview of what we have considered.

- Vanilla GANs or DCGANs and other classic GANs are not ideal for controlling the topography, since they are not built for the direct image translation. However, if so happens that the simulation does not have obey a user-controlled topography requirement these are capable of producing reasonable quality samples produced via unconditional generation. Some visual examples generated with DCGANs are presented in Figure 3.3.

- CycleGANs - are the prime example of image translation models for unaligned datasets. This means that having two datasets without direct correspondence between them we could "translate" the style of the source image into the target image style. That spares the time for creating semantic maps, or labelling data in any other fashion.

  In theory the translation of some simulated imagery into realistic looking sonar scans could be possible. It so happens that Seebyte, the company that kindly provided us with the sonar data for this chapter, also has an in-house moderately realistic artificial simulator called SonarSim, briefly used in [123]. We obtained a bunch of images from SonarSim, even balancing out the types of terrains observed in both datasets for maximal resemblance. The results unfortunately were not satisfying - looking more like an overlay of some kind over the original source images. Some examples of the achieved results are provided in Figure 3.4

- pix2pixHD - a version of pix2pix that works for higher dimensional (2048×1024 pixels) image translation. On its own it would not be enough for generation of

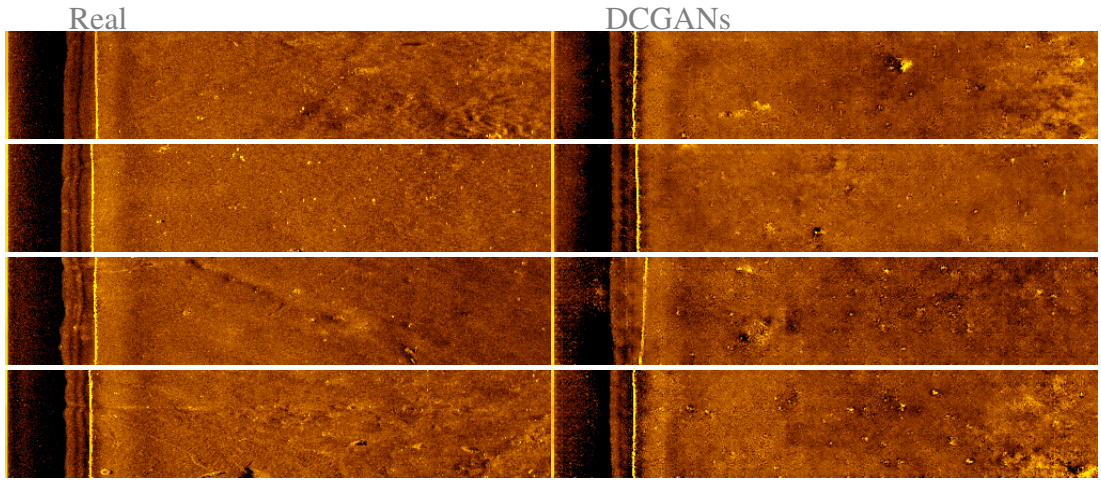

Fig. 3.3 **DCGANs visual results:** all the images on the left are real, all the images on the right are generated. DCGANs generate images at random, simply sampling the underlying distribution of the training set of images provided. Hence, the horizontal pairs of images are for general qualitative comparison, but are not directly related.

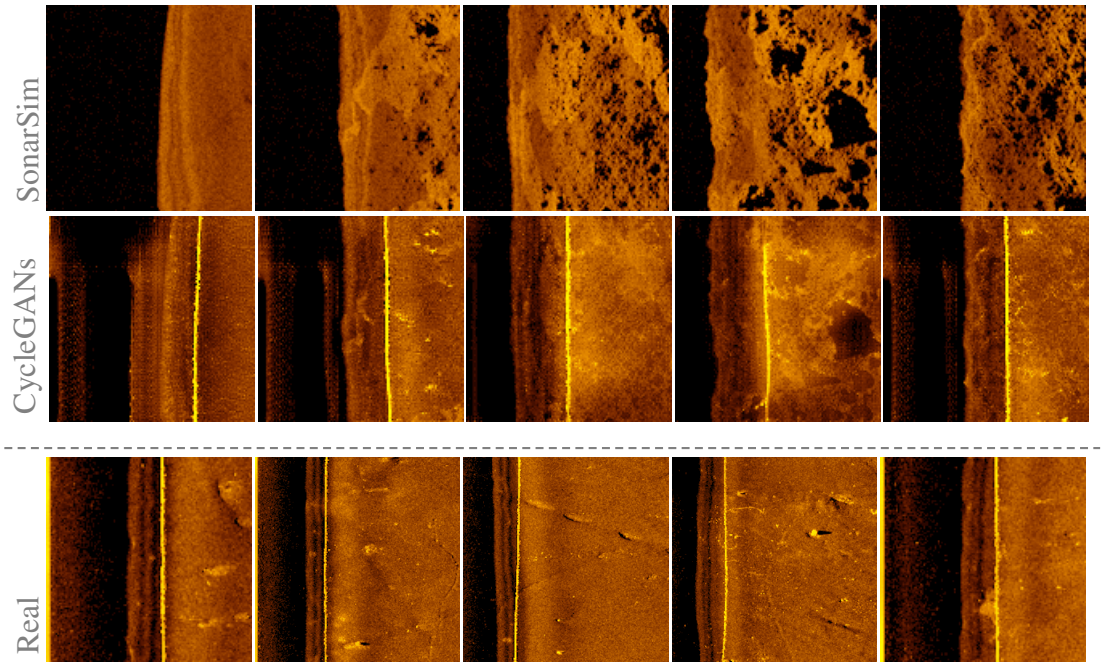

Fig. 3.4 **CycleGANs visual results:** top row corresponds to slightly artificially looking images produced by the SonarSim simulator, second row is CycleGANs attempt to translate the above imaged into something more realistic. Third row bears no direct correspondence to the other two - it is only provided for the rough quality comparison. Unfortunately the synthetic images generated by CycleGANs are not of sufficient quality, featuring a lot of inconsistencies and unnatural artefacts.

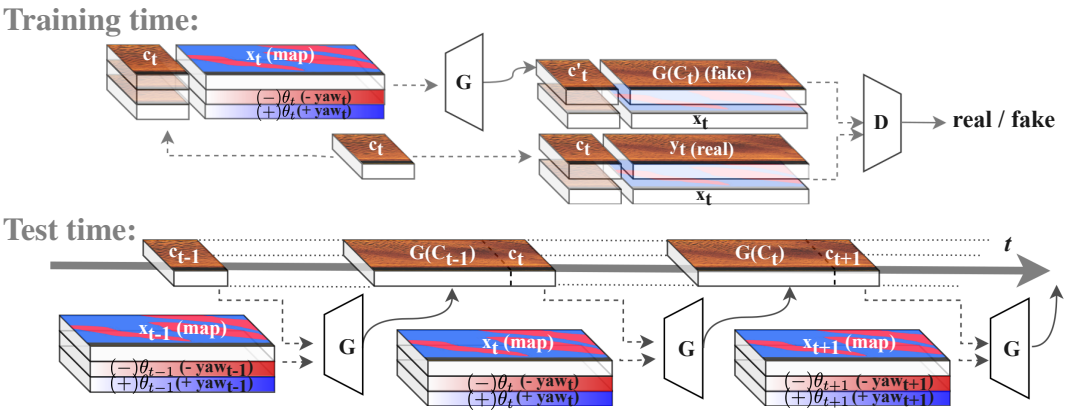

Fig. 3.5 **Training time:** similarly to the pix2pix, the generator inputs semantic maps corresponding to the desired topography x_t, outputs synthetic sonar-scan data $G_t(C_t)$. It is extended to accept two additional conditions - a snippet of the previous image c_t facilitating continuity in the generated mission and yaw indicating the requested turns of the vehicle. Output is then labelled by discriminator as real or fake along with the real images. **Test time:** at each time-step, the generator processes a semantic map of requested configuration, yaw variable (responsible for turn distortions, defined by the vehicle trajectory), and a small snippet of the previous synthetic image to enforce the continuity of the seabed throughout the mission.

whole missions worth of data, which is 300000×1024 pixels minimum, however it could be adapted in a similar fashion as vanilla pix2pix. We do not use it purely because of the training data privacy limitations vs. computational power requirements for this method[2]. However, if the need will be the architectural changes for extending the pix2pixHD into the unlimited resolution generator would be identical to what we did for the classic pix2pix.

3.4 Markov-Conditional Pix2pix

In this section we are going to explain the principle of exploiting the Markov assumption for unlimited sequential generation of the image fragments in the along track direction (in the direction of the moving vehicle), resulting in continuous extremely high-dimensional image. The resulting method is capable of generating smooth and continuous extremely high dimensional images, equivalent to full mission worth of sonar data.

[2]The data are under NDA, and under no condition should leave the office, where the best GPU available is limited to 12GB RAM. The official specs of pix2pixHD state the requirement of two GPUs, 24GB RAM each, to reproduce the published results

3.4.1 Training Data

The real side-scan sonar data used for the experiments is acquired with a Marine Sonic sonar. Its across-track resolution is 512 pixels ($\times 2$ for both port and starboard channels). The vehicle travels at the speed of 1 meter per second and generates approximately 16 pings per second. As the vehicle turns it causes distortions in the images, and models that generate synthetic imagery should be expected to produce similar distortions. Our model accounts for these distortions using the desired vehicle attitude information (yaw, pitch, and roll). Only yaw information is provided by the gathered training data, but we note that our method is able to incorporate pitch and roll data as well.

To create a training dataset, sonar scans were sliced into non-overlapping 464×512 images y_t, we also have the corresponding semantic maps x_t, and they are sliced in the same way (464×512). The training images (and maps) can be overlapping if the data availability is an issue, however this should be done carefully as repetition might encourage pattern memorisation by the network. In addition to the training images we use slices of the previous adjacent images c_t as conditions facilitating the continuity of the final result. In our case they are of size 20×512, but this was an arbitrary selection, and can be changed depending on the resolution of the training data. Please note that (c_t, x_t) concatenated together give us $512 \times 512 \times 1$, which overlayed with two more channels of yaw-based variable θ_1 (explained in Conditions subsection below) gives $512 \times 512 \times 3$ - the size of the input for MC-pix2pix networks, as explained in Figures 3.6 and 3.7.

Our model was trained on a relatively small dataset of 540 of these non-overlapping images (and their corresponding semantic maps). Increasing the training set size might bring further improvements but in our experience this method works with as few as 200 training image samples.

3.4.2 Assessment metrics

In addition to the visual examples provided in Figure 3.8, the model performance has also been quantitatively assessed using the following metrics:

- Human visual assessment score: we provide the statistics on distinguishing real sonar imagery from the synthetic images. It is collected from 30 participants with a variety of experience of working with underwater sonar data. During the test, participants were allowed to inspect images without the time limit.

- Freschét Inception Distance [3]: often used for quality assessment of generated images, this is a heuristic for measuring the difference between the real and synthetic image distributions, explained in more detail in Subsection 2.1.5.
- Generation speed at test time: crucially for practical application, generative models must provide results of the requested quality without compromising the speed of generation. A minimal requirement is an order of magnitude faster than real data collection speed.
- Performance improvements of a bounding-box-based ATR detection algorithm when bootstrapped with the synthetic data along with the available real data. This is assessed in terms of mean Average Precision (mAP) and F1-score - harmonic mean between the precision and the recall of the ATR system. The intersection over union (IoU) for bounding boxes is set relatively low at 0.4. The ATR assessment metrics are calculated in the following way:

$$\text{True Positive} = (IoU > 0.4)$$

$$mAP = \frac{1}{N}\frac{1}{C}\sum_{C}\sum_{N}\left(\frac{True\ Positives}{True\ Positives + False\ Positives}\right)$$

where N is the number of teset targets of one type, and C is the average across the types of "contacts" - objects of interests.

$$F_1 = \frac{True\ Positives}{True\ Positives + \frac{1}{2}(False\ Positives + False\ Negatives)}$$

3.4.3 Method and Architecture

A top-level overview of the model architecture is provided in Figure 3.5[3]. It resembles the fully-convolutional pix2pix architecture [56], as explained in section 2.1.4, resized for $512 \times 12 \times 3$ input. Please refer to figures 3.6 and 3.7 for more details on the exact networks specifications, please also refer to the rest of the current section 3.4.3 for more specific details on training this model.

Importantly, this model is designed to accept two conditions at the input level [1], these are being concatenated to the semantic maps and submitted as a single input to the generator and the discriminator as an input.

[3]Please note: semantic maps, sonar-scans, and yaw variables are all single-channel and are only coloured as RGB for illustration purposes.

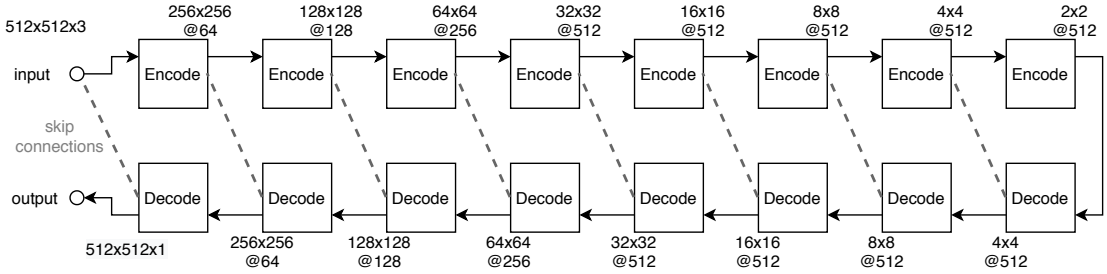

Fig. 3.6 **MC-pix2pix generator network:** as mentioned before our method is based on pix2pix, and the generator has Unet architecture. The generator inputs monochrome semantic maps corresponding to the desired topography x_t, concatenated with a snippet of the previous image c_t and overlayed with yaw, resulting in a $512 \times 512 \times 3$ input.
It is then being pulled through 8 convolutional layers, each of them with kernels of size 4, stride 2, and padding 1, with batch normalisation and activated with leaky ReLu (negative slope 0.2).
After that 8 deconvolutional layers follow - each features batch normalisation, Unet skip connection block. All but the last one have ReLu activations - the last one is equipped with Tanh activation function. They also have kernels of size 4, stride 2, and padding 1. The output is $512 \times 512 \times 1$ monochrome sonar image.

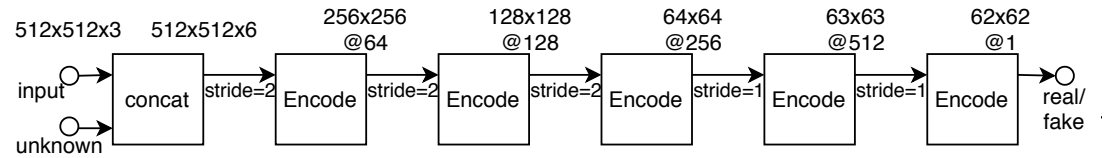

Fig. 3.7 **MC-pix2pix discriminator network:** discriminator is very similar to the pix2pix discriminator presented in section 2.1.4 in figure 2.6.
First of all, the discriminator inputs the generator input - monochrome semantic maps corresponding to the desired topography x_t, concatenated with a snippet of the previous image c_t and overlayed with yaw, resulting in a $512 \times 512 \times 3$ input.
Secondly it inputs a corresponding realistic image - either from the real training set or produced by the generator. It concaternates the input and puts it through 5 convolutional layers, each of them with kernels of size 4 and padding 1,, and stride 2 (changing to 1 for the last two layers). Each convolutional layer is followed by a batchNorm and is activated with leaky ReLu (negative slope 0.2).
The output is the decision on whether or not the unknown part of the discriminator input is real or generated.

Conditions

The first condition, c_{t-1}, enforces the visual continuity of the generated output at test time. The information is conveyed by a short snippet taken from the end of the previous image, enabling the model to run self-conditionally at test time, as illustrated in Figure 3.5.

The second condition is an empirically derived yaw-based metric θ_t, that takes care of the image distortions caused by turns, it is calculated as:

$$\theta_t = 5 \;\; max(1, |\psi_t - \psi_{t+50}|) \tag{1}$$

where ψ_t is the yaw for ping t, and the sign of $(\psi_t - \psi_{t+50})$ is used to determine whether the clockwise or counterclockwise turn is expected. The coefficient 5 is derived empirically, based on the visual quality of the generated samples, featuring turns. θ_t is calculated per ping (per row) of the corresponding semantic map x_t, the resulting vector gets repeated column-wise, separated into two arrays based on the sign of $(\psi_t - \psi_{t+50})$, and overlayed with the single-channel semantic map x_t, completing the generator input.

At training time:

The generator inputs the single-channel semantic maps x_t and the two conditions - yaw variable θ_t and the previous image snippet c_t. The generator outputs single-channel generated sonar images $G_\gamma(x_t, ...)$. The generated images are non-overlapping across the time-steps, however each consecutive generated image is based on the previous one due to conditioning on the previous image snippet c_t.

The discriminator receives all available data except the yaw variable - semantic maps x_t, condition c_t, and real images y_t, and generated images $G_\gamma(x_t, ...)$. The discriminator outputs the verdict on whether the image is real or fake. The discriminator is rewarded based on how well it can distinguish the synthetic image $G_\gamma(x_t, ...)$ from real y_t, generator - based on if it managed to "fool" the discriminator.

This model is adversarially trained with Adam optimiser (learning rate 0.0002 and $\beta_1 = 0.5$, as in classic pix2pix) for 200 epochs, with batch-size of 10, and 3 repetitions of discriminator for 1 of generator per each epoch with the following loss function:

$$\begin{aligned} G_\gamma^\star = \arg\min_{G_\gamma} \max_{D_\phi} \big\{ & \mathbb{E}_{x_t, y_t}[logD_\phi(x_t, y_t)] \\ + \;\; & \mathbb{E}_{x_t, \mathbf{C}_t, z}[1 - logD_\phi(x_t, G_\gamma(x_t, \mathbf{C}_t, z))] \\ + \;\; & \mathbb{E}_{x_t, \mathbf{C}_t, y_t, z}[\big\| y_t - G_\gamma(x_t, \mathbf{C}_t, z) \big\|_1] \big\} \end{aligned} \tag{2}$$

where x_t are semantic maps, y_t are real sonar images, z is random noise vector, and $\mathbf{C}_t = [c_{t-1}, \theta_t]$ is a collection of condition variables for the generator, and γ and ϕ are the specific parameters of the generator and the discriminator networks correspondingly. The first two lines of (2) represent the discriminator and generator losses respectively, and the last one is the L1 loss, a regularization term that is meant to discourage blurring in the generator output [56].

At test time:

Only the generator is used and runs identically to the train-time, except c_t now comes from the end of the previous image generated.[4] The model output is therefore dependent on its own previous output and capable of producing consistent and continuous images of any length.

3.4.4 Experiment 1: Image quality assessment results

In this experiment we compare MC-pix2pix with real images as well as with the output of the original pix2pix and pix2pix with post-processing, i.e., with blending the borderline between the separate synthetic snippets using sigmoid-function smoothing. The achieved results are compared both qualitatively and quantitatively as follows:

Visual examples

Visual examples of all the methods are provided in Figure 3.8. These are directly comparable as they are generated from the same semantic maps (left), obtained via segmenting the real seabed images (right). Their underlying generative models are trained for the same number of epochs on the same dataset. In order to further eliminate the disadvantage for baseline methods that do not use the yaw variable, no yaw variation was applied in this example (i.e., no turns). This example is primarily illustrating the consistency of the MC-pix2pix output compared to the baselines.

Visual assessment scores

Visual assessment scores are obtained from 30 human experts (different levels of experience with sonar data - from introductory course to several years of work with sonar images). Although it is common to use Amazon Mechanical Turk for obtaining such

[4] The very first condition c_0 is an exceptional case, since there is nothing to draw this zero-time condition from. This situation can be resolved by either using a snippet of some real seabed image (non-generated), or even using a random noise snippet. In the second case we recommend to crop off the first few rows of the very first generated image, as they will not be of good quality. However the rest of the first generated image conditioned on a noise snippet will be perfectly realistic.

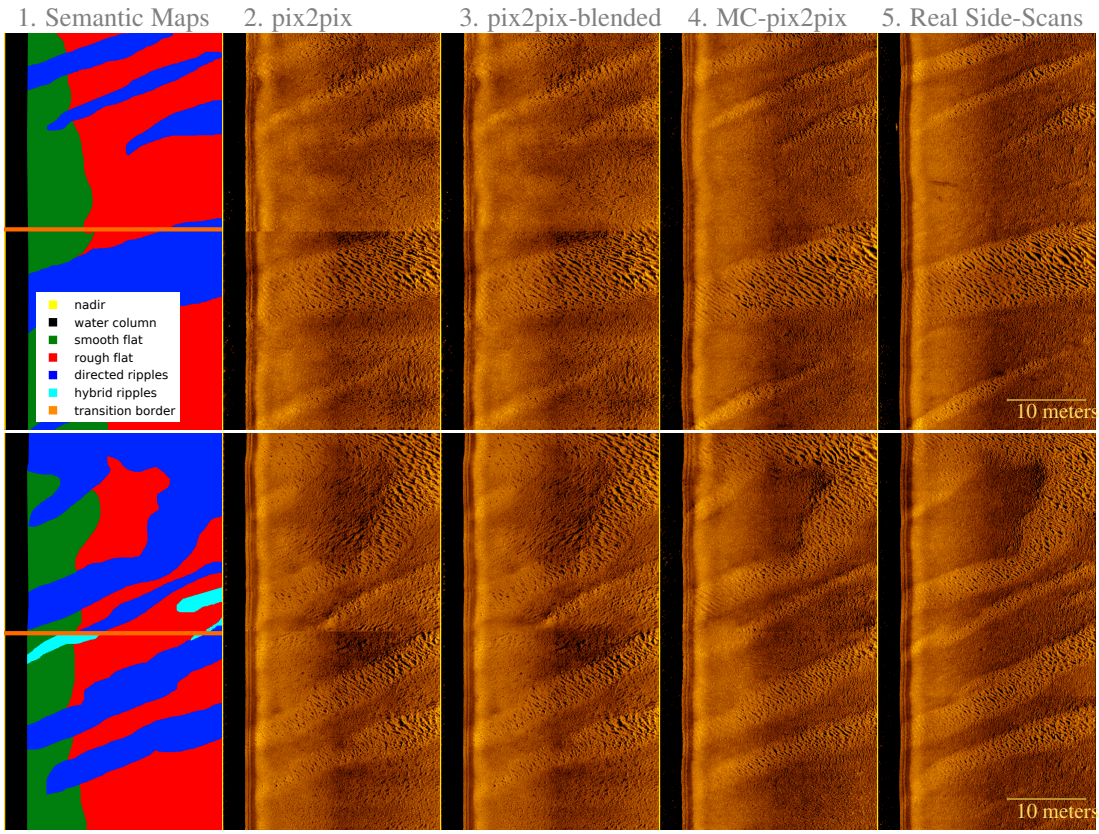

Fig. 3.8 **Visual Comparison (left-to-right): 1.** Semantic maps are used as an input by all of the compared models. In reality the semantic maps are grey-scale with different shades corresponding to different types of terrain. Colour is introduced for visualisation purposes only. Border-line label indicates the transition between the images for convenience of the reader and is not present in the input of the model. **2.** Original pix2pix example has a clear sharp transition border between the images (in the middle). This is because the image patterns or intensities are not shared between adjacent images. **3.** pix2pix with sigmoid-smoothing applied at the transition demonstrates that simple post-processing is not particularly good at matching the textures of the seabeds. **4.** MC-pix2pix (ours) has clearly smoother transition border, enabling it to produce continuous imagery for missions of any length when run repeatedly. **5.** The real data example.

Metrics	pix2pix	sigma-pix2pix	MC-pix2pix
Mean accuracy of labeling	0.64 (±0.22)	0.62 (±0.18)	**0.52** (±0.11)
Synthetic labeled as real	0.34 (±0.28)	0.42 (±0.24)	**0.54** (±0.17)
Mean time per image (sec)	4.85 (±2.63)	4.86 (±2.60)	**6.13** (±3.23)
Freschét Inception Distance	0.9257	1.0241	**0.7834**

Table 3.1 **Image Test Scores:** the average accuracy around 0.5 shows that humans are as good as random at telling MC-pix2pix images apart from real. MC-pix2pix gets labelled as real more than the competitors and comes the closest to the 0.66 ratio of real images labelled as real. Image processing times show MC-pix2pix images are the most challenging to inspect. The lowest FID score confirms the MC-pix2pix is closer to the real image distribution than the competitors.

assessment, it is not feasible in our study since real data are both commercially sensitive and too specialized to get a valuable assessment by people previously unexposed to the sonar imagery. Instead we obtain our assessments from the human experts who possess some knowledge of the domain.

The test consists of a number of images generated by MC-pix2pix, pix2pix, sigmoid-blended pix2pix, and corresponding real examples presented in even proportions for a human expert to classify as real or synthetic. The order of images from different models is randomised to avoid putting any of the methods into a disadvantage of being examined last. The total amount of images assessed by a single expert during the visual assessment is 20 per method + 20 corresponding real images for comparison so 20×4, which results in 80 per person.

Results are presented in Table 3.1. Domain experts had 0.52 mean accuracy labelling MC-pix2pix images as real or fake. This is essentially an optimal result because for a two-class problem (real or fake), proximity to 0.5 means experts being as good as random at telling the synthetic data apart from real. Further we present the proportion of synthetic data mislabelled as real (i.e., the success of generator in “fooling" human experts). For comparison, the proportion of the real data labelled as real is 0.66. The last metric of the visual assessment is the average time taken to make a decision on a sample. Interestingly, participants spend more time on MC-pix2pix images, which suggests these were more challenging to classify. Our method compares favourably to all of the presented baselines for all the presented metrics.

Test: Flat	Real only	+ Noise	+ SonarSim	+ MC-pix2pix
mAP	0.30	0.39	0.27	**0.45**
F1-score	0.57	0.58	0.50	**0.60**
Test: Complex	Real only	+ Noise	+ SonarSim	+ MC-pix2pix
mAP	0.00	0.01	0.00	**0.11**
F1-score	0.23	0.59	0.62	**0.68**

Table 3.2 **Bootstrapped ATR performance improvement**: MC-pix2pix improves ATR mAP and F1 score compared to just using real data, as well as beats the baselines for both non-complex flat (top) and complex (bottom) test terrains.

Freschét Inception Distance (FID)

FID is also provided at the bottom of the Table 3.1. Lower values correspond to the distributions closer to the real one. For instance the FID between two real data sets would be approximately zero, whereas the FID between a constant and $U(0,1)$ is greater than 6. FID is calculated on test images of size 1856×512. This size was chosen arbitrarily - similar results are expected from larger images. FID is sensitive to scaling, so the data for FID assessment has been normalized to the [0, 1] range.

Generation speed

The MC-pix2pix is expected to be fairly close to the original pix2pix in speed due to our model being an extension of pix2pix. We used GTX 1080 Ti (12GB RAM) for estimating the MC-pix2pix generation speed. MC-pix2pix is approximately 18 times faster at test time than the real acquisition speed.

Marine Sonic ($512\times n$ resolution sonar) speed of data acquisition is about 15 000 pixels per second depending on the settings. MC-pix2pix speed of generation at test time is about 274 690 pixels per second with image loading/processing (18.31266 times faster than the real speed of acquisition) and about 304 819 pixels per second (20.32127 times faster than the real speed of acquisition) excluding the image loading time. This is for GTX 1080 Ti (12GB RAM), with MC-pix2pix implementation in Pytorch.

3.4.5 Experiment 2: Improving ATR training with MC-pix2pix

Motivation

Training ATR on simulated data is useful in case of the lack of complexity in training data, or the lack of training data itself, in which case adding more realistic simulated

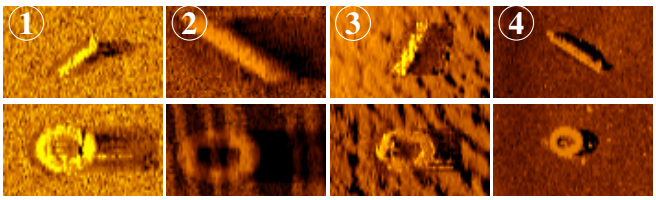

Fig. 3.9 **Examples of the two target objects (tyres and cylinders) that we used for the ATR tests:** 1. Random uniform noise background, 2. SonarSim[2], 3. MC-pix2pix, and 4. Real data. Objects in pictures 1-3 are synthetic, inserted with the [123][3] method.

data would be beneficial. If certain seabed types are underrepresented in the currently available training set, but MC-pix2pix was exposed to these types of terrains before, it can enrich the dataset with additional seabed types.

Experiment goals and the ATR network

In both of these cases we check the increase in the ATR performance between training on just a small real dataset and enriching it with MC-pix2pix. We assess the performance with mAP and F1-score, as explained in Sec. 3.4.2. We are interested only in the increase of the ATR performance facilitated by GAN-generated data - the ATR performance level itself is irrelevant here, we also do not claim any novelty or merit behind the ATR architecture itself, it is borrowed from the original papers [125, 9].

We are using a very simple template ATR mechanism. It is bounding box based with the intersection over union (IoU) between the predicted and the ground truth bounding box being set rather low at 0.4, because of the low quality and quantity of the available annotated training examples, where annotations are neither precise nor consistent. Hence, considering the sparsity of the annotation, the main focus of the ATR here is detecting something at all for further inspection.

The network used here is a generic RetinaNet-type network [125, 126], trained with focal loss, with ResNet-18 (18 layers) backbone [9], i.e. ResNet-18-FPN, implemented exactly like in the original paper. The only modification made to this template architecture is that the number of filters is halved throughout the architecture compared to the original paper 3.10 for the simplified explanation of the RetinaNet structure, and Figure 3.11 for the ResNet-18.[5] It is trained with Adam optimiser over 25 epochs.

The training data has 300 objects of interest (150 cylinders, 150 tyres) per type of seabed (such as real, noise, SonarSim, and MC-pix2pix/R2D2). The test data also has the same amount of objects.

[5]The entire figure credits are [125] and `https://www.pluralsight.com/guides/introduction-to-resnet`

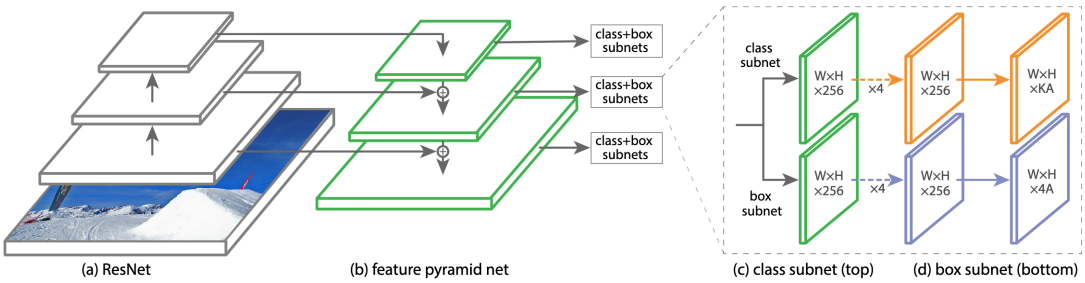

Fig. 3.10 **RetinaNet network architecture** uses a Feature Pyramid Network (FPN) on top of a feedforward ResNet architecture [9] (a) to generate a rich, multi-scale convolutional feature pyramid [126] (b). To this backbone RetinaNet attaches two sub-networks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d).[5]

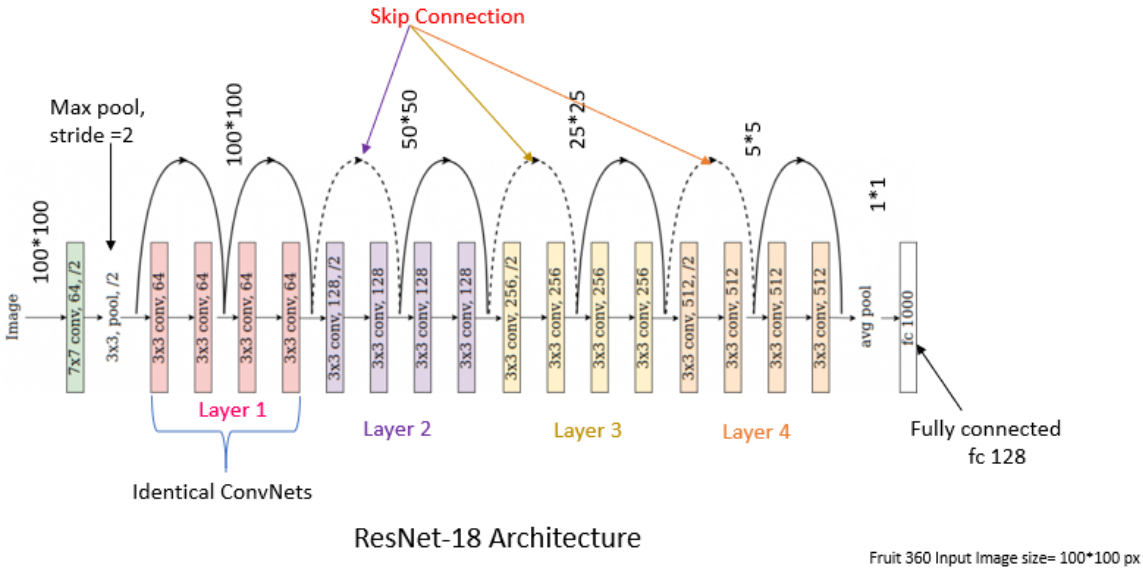

Fig. 3.11 **The architecture of the ResNet-18**, that gets used as the part of the RetinaNet, as appears under (a) in Figure 3.10.[5]

Baselines explained

We train 4 ATR networks on the corresponding datasets: real data only (flat and non-complex), the same real data plus MC-pix2pix images, and baselines - real dataset plus uniform random noise backgrounds, and real dataset plus SonarSim seabeds[6]. All except the real data are augmented with synthetic targets using the method from [123][7], examples of these are provided in Fig. 3.9, as per images presented there, we used two types of the objects - cylinders and tyres.

Experiment 2.1: Data shortage

For this experiment MC-pix2pix was trained on the available real training set (flat and non-complex). The MC-pix2pix-generated data were used to train the ATR, which then was tested on another flat and non-complex dataset. Table 3.2 (top) shows that MC-pix2pix provides significant improvements in MAP, compared to just real data and other baselines, and the best F1.

Experiment 2.2: Lack of complexity

In this case MC-pix2pix was pre-trained with slightly more complex ripply seabeds, emulating previous exposure to the complex data. It then generated more of the complex seabeds, that were used to train the ATR alongside the flat and non-complex real data. When testing this ATR on complex real terrains (results presented at the bottom of Table 3.2), both F1-score and mAP drastically improve with the MC-pix2pix data bootstrapping, compared to just real data training and baselines.

This confirms that MC-pix2pix could be deployed as a highly efficient bootstrapping technique for improving ATR performance in cases of low real data availability or low real data diversity, that are common in the real life applications.

[6] SonarSim - standard vaguely realistic side-scan simulator as used in [123], capable of generating various seabed textures with limited user control over the type of generated data, but not the exact topography.

[7] Due to extremely low amount of the training data for targets we could not generate targets with MC-pix2pix.

3.4.6 Addressing Potential Concerns

Advantages compared to generating the seabed piece-wise and then stitching it together with post-processing

Although standard smoothing techniques like sigmoid-smoothing interpolate well between the colours, they do not conduct the texture integration between the images as is evident from the Fig. 3.8 middle section (at border-line).

Quality Decay over the generation time

Self-conditional model at test time suggests that the quality drop could accumulate over time. However, the nature of GAN architecture prevents this - trained GAN always samples the training data distribution, regardless of the condition. This has been verified via generating a standard test mission - average duration of 2 hours, 300,000 pixels along the track.

Handling vehicle turns

We condition on the yaw only (because roll and pitch are not available in our data). It does not seem to be quite enough information to make results fully realistic, however the model not only acknowledges the concept of a turn distortion but also distinguishes between the inside and the outside turns, in some cases producing very realistic results (especially successful for the outside turns simulation). A visual example of what real vs. generated turns look like is presented in Fig. 3.12.

Handling multiple viewpoints

Typically a vehicle observes the same area at least twice. The MC-pix2pix accounts for the topographical coherence with respect to the terrain types. The model naturally produces a similar image for the different viewpoints. Example in Figure 3.13 shows two views of the same area (e.g. the image synthesized for the same map approached from different sides) and their overlay.

Unconditional generation

This work focuses on producing the missions with user-controlled topography, however if one needs to avoid specifying it (e.g. when producing the training data for ATR) the original GAN [4] and modifications, such as DCGANs [2], can be employed to generate some semantic maps for the input into MC-pix2pix.

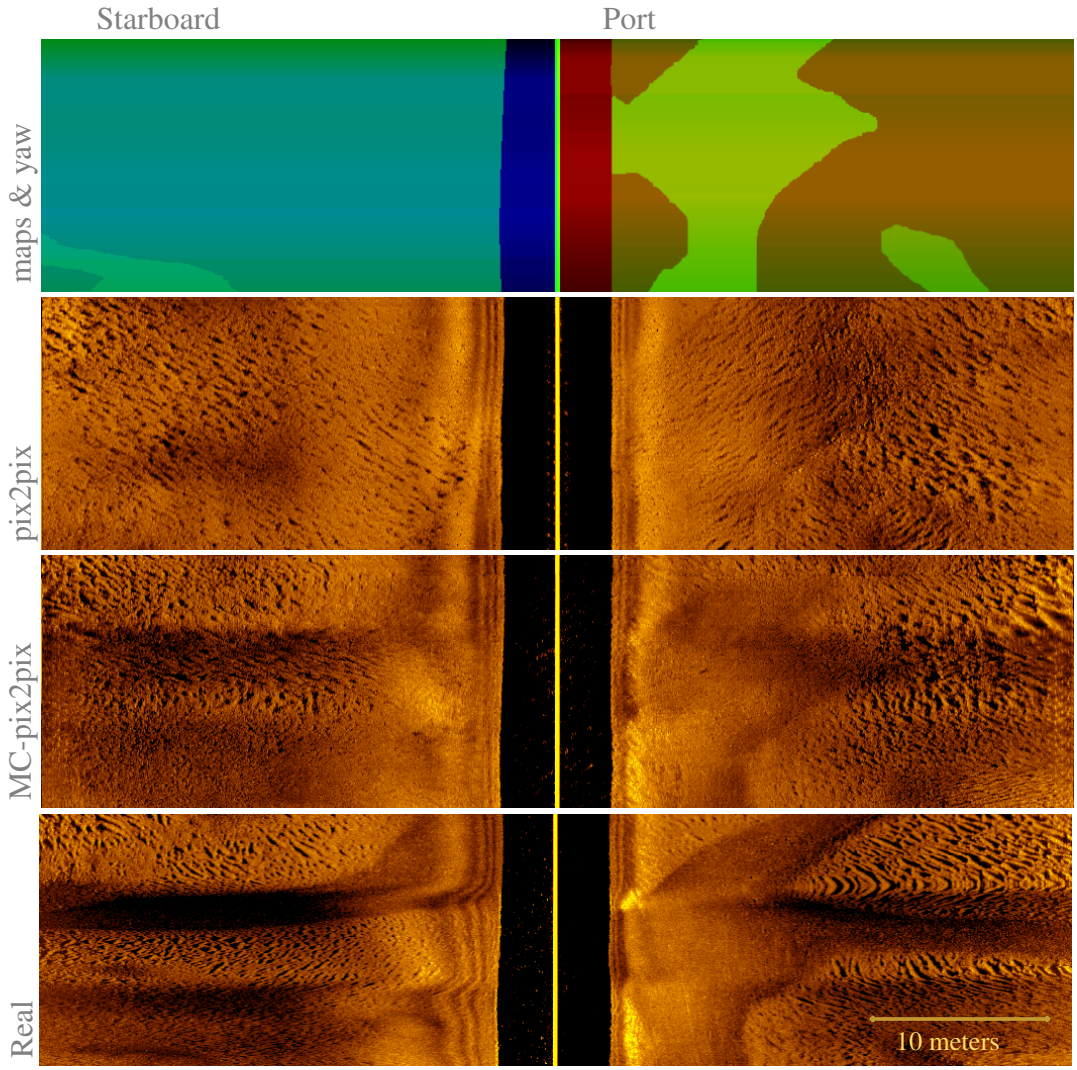

Fig. 3.12 **Reproducing turns with yaw-conditioning (top-to-bottom):** 0. semantic map overlayed with yaw. Semantic maps for the training set get labeled in RGB for the convenience of the human labelling, then translated into grayscale, placed in green channel, then yaw variables get duplicated in blue and red channel, with sign corresponding to the direction of the turn. Hence the resulting discoloured green shade image with red and blue overlay. 1. pix2pix fails to capture a distortion caused by vehicle turning as semantic map provides no indication of turns other than perhaps indirectly through the topography. 2. MC-pix2pix benefits from a inbuilt yaw-based condition, getting close to real turn patterns and distinguishing between the inside (left) and the outside (right) turns. 3. Real example - in this case much sharper distorted image compared to the others. Real turns come in wide variety, so even the expert operators are never certain which are real and which are simulated. During the real data processing and analysis by human operators turns are almost always discarded, since they are often confusing. More modern vehicles completely switch off the sonars at the turns assuming that the data gathered will not be informative.

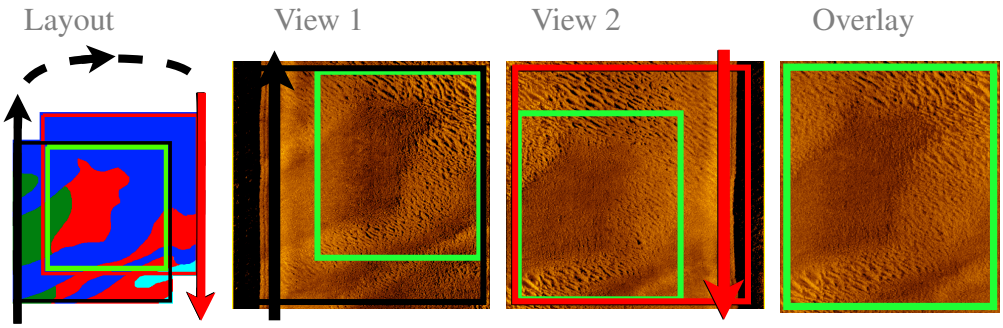

Fig. 3.13 **Handling different viewpoints:** MC-pix2pix is topographically consistent with respect to the types of terrains. This example shows the same region, generated as perceived from two different viewpoints with some displacement. Generated images show a clean overlap without contradictions.

Other potential applications

MC-pix2pix can be potentially used in any setting where simulation of large-scale continuous data is required. Useful for bootstrapping learning algorithms, environment feature detection and recognition, or even for side-scrolling games background generation.

3.4.7 Conclusions

This section proposes a method for generating realistic synthetic sonar sensory data for full-length underwater missions with a direct control over the topography. To our knowledge this is the first published work that addressed the generation of side-scans for entire missions with generative adversarial networks. This method results in an immediately applicable for visual simulation of sonar data (at classic MarineSonic resolution). Both visual and quantitative results were provided in this section. They show that the generated images are both indistinguishable from real by human experts and capable of boosting the performance of the ATR systems. An additional advantage of this recursive architecture is that it is relatively undemanding about the hardware. It is being supported even for the GPUs with very modest RAM capacities, which is almost never an option for the other higher-resolution varieties of GANs.

This work can be extended further by investigating the use of roll and pitch data for improving the quality of the simulation for the vehicle turns, subject to the availability of the suitable training data[8].

However, the main extension to this work is generation of the data for higher fidelity sonars, such as EdgeTech (an order of magnitude higher resolution compared to the

[8]Unfortunately our training dataset only had yaw available.

Marine Sonic data presented in this work), or SAS sonars (two orders of magnitude higher in resolution compared to Marine Sonic). Despite being very challenging this problem can be addressed with some limited extensions to the current MC-pix2pix method. These are presented in the next section.

3.5 R2D2-GANs for Unlimited Resolution Image Generation.

In this section we are going to extend the previously introduced MC-pix2pix principle for sequential generation along the two dimensions (both x and y). Having the ability to generate any length of mission via MC-pix2pix, we are now exploring the ability of generating images for any resolution of a sonar as well. Results presented in this section simulate the output of the EdgeTech - sonar, that is 4620 pixels across the track (as compared to 512 pixels across the track for Marine Sonic, used for illustrating the work of MC-pix2pix in section 3.4).

We call this method double-recursive double-discriminator Generative Adversarial Networks (R2D2-GANs, or “R2D2" for the sake of conciseness). To our knowledge, this is the first technique capable of adversarial generation of continuous and realistically-looking sonar side-scans of any requested size or resolution.

Potential applications of R2D2 can go far beyond the sonar imagery, as it can produce any type of large resolution imagery, provided a sufficient amount and quality of the initial training examples.

The visual examples of the results of the R2D2-GANs are provided in the Figure 3.17. Results demonstrated in this work are acquired with the image-to-image translation based architecture [56], which could be easily altered to accommodate another type of GAN, in order to better facilitate different simulation objectives.

3.5.1 Method and Architecture

The R2D2 also belongs to the family of GANs. More specifically the R2D2 is just an extension of the Markov-conditional image-to-image translation technique, MC-pix2pix [6], re-purposed to enable image generation in larger resolutions, and hence also based on pix2pix [56] , which has been explained in 2.1.4. The specific networks used for the generator and both discriminators in R2D2 are of exactly the same architecture as in MC-pix2pix, Figures 3.6 and 3.7. It is the different way the input (and output)

themselves are constructed[9], that combines these network architecture into a different top-level model, which is provided in figures 3.14 and 3.15.

There are multiple conditional flavours of the basic GAN architecture. Henceforth we focus on the paired image-to-image translation techniques, of which the pix2pix [56, 58] is a prime example. This choice is dictated by the requirement of the user-controlled topography.

The main features of the R2D2 technique:

(i) incremental recursive generation (extending the Markov principle from [6]) applied along two axes (rather than just one, like in [6]). This allows for handling any across track resolution.

(ii) an additional discriminator is introduced for the coherence control of resulting larger scale images. Figures 3.14 and 3.15 provide the schematic illustration of the proposed method.

At training time:

As per scheme in Figure 3.14, first, the larger training images and their semantic maps are partitioned into multiple tiles. The generator inputs noise and the semantic maps of the current tile - for topography control. It also uses the additional conditions that include the location of a pixel in the across track direction and small snippets taken from the adjacent tiles above and to the left of the current tile. These are to ensure the continuity of the resulting large image. The generator is trained to produce realistic images given the above inputs and conditions.

The discriminator D_{1_ϕ} then tries to distinguish the real imagery from the simulated. The discriminator D_{2_ψ} does the same, but processing the larger images (2×2 tiles) with the newly generated tiles embedded into them, and tries to distinguish them from the real unedited larger images.

The results provided in this chapter are building upon the fully-convolutional pix2pix-style architecture with 9 resnet blocks [56], as explained in 2.1.4, extended with additional conditions to support incremental recursive generation and additional “bigger picture" discriminator D_{2_ψ} to further encourage the smoothness and continuity of the resulting image. This model is adversarially trained with Adam optimiser (learning rate 0.0002 and $\beta_1 = 0.5$, as in classic pix2pix), for 10 epochs with batch-size 3, and

[9]Conditions from adjacent tiles and semantic maps (c_1, c_2, x_t) concatenated together as a single $512 \times 512 \times 3$ input for G and D_1, and larger tile X_t (scaled down to 512×512) as D_2 input, and corresponding expected outputs.

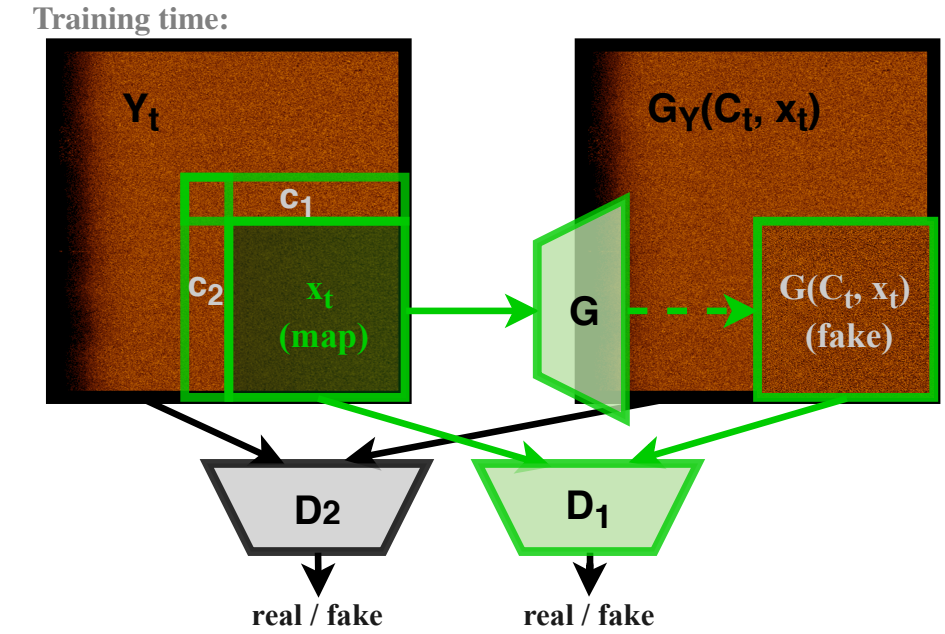

Fig. 3.14 **At training time:** inputs of the generator network include conditions c_1 and c_2, which are small snippets of adjacent tiles (top and left) of the currently generated tile. The output of the generator G is the suggested synthetic image tile, generated taking into account the input conditions. The output of the generator is then assessed by the first discriminator D_{1_ϕ} along with real tiles. The second discriminator D_{2_ψ} assesses the larger real image variants (2×2 tiles) - the unchanged and the edited with a generated tile. Both discriminators issue their decisions on whether the image is real or synthetic, and are rewarded based on the correctness of their decisions. Losses of both discriminators are then back-propagated through to the generator and used for the adversarial training.

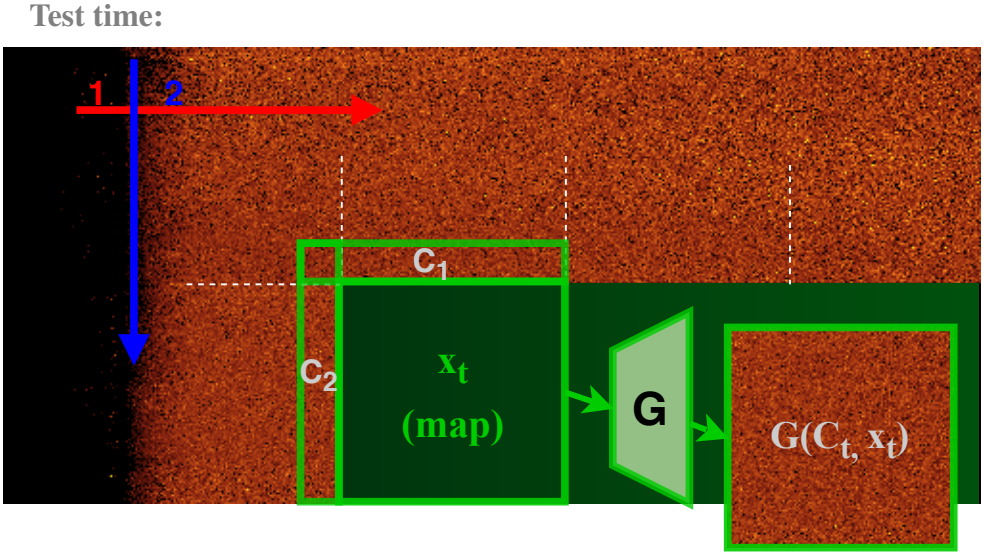

Fig. 3.15 **At test time:** only the generator is used at this stage. It produces image tiles first left-to-right, and then top-to-bottom. Each tile is conditioned on adjacent image snippets of the tile above and to the left of the currently generated tile. These conditions help to maintain the continuity of the larger picture produced at test time.

3 gradient updates of discriminator D_{1_ϕ} corresponding to each gradient update of the generator. The training loss function can be summarised as follows:

$$
\begin{aligned}
G_\gamma^\star = \arg\min_{G_\gamma} \max_{D_{1_\phi}, D_{2_\psi}} \Big\{ & \mathbb{E}_{\mathbf{C}_t, x_t, y_t, z}[\big\| y_t - G_\gamma(\mathbf{C}_t, x_t, z) \big\|_1] \\
+ \frac{1}{2} \Big(& \mathbb{E}_{\mathbf{C}_t, x_t, y_t}[logD_{1_\phi}(\mathbf{C}_t, x_t, y_t)] + \mathbb{E}_{X_t, Y_t}[logD_{2_\psi}(X_t, Y_t)] \\
& + \mathbb{E}_{z, \mathbf{C}_t, x_t}[1 - logD_{1_\phi}(\mathbf{C}_t, x_t, G_\gamma(\mathbf{C}_t, x_t, z))] \\
& + \mathbb{E}_{\mathbf{C}_t, z, X_t}[1 - logD_{2_\psi}(X_t, Y_t(G_\gamma(\mathbf{C}_t, x_t, z)))] \Big) \Big\} \quad (1)
\end{aligned}
$$

where x_t are semantic maps and y_t are real sonar images per tile. X_t and Y_t are larger (2×2 tiles) semantic maps and sonar images respectively. X_t and Y_t include tiles x_t and y_t correspondingly. z is a random noise vector, and $\mathbf{C}_t = [c_1, c_2]$ are the condition variables for the generator. $Y_t(G_\gamma(\mathbf{C}_t, x_t, z))$ stands for a larger image Y_t (2×2 tiles), where the native tile y_t is replaced by the generated tile $G(\mathbf{C}_t, x_t, z)$. And γ, ψ, and ϕ are the individual network parameters for the generator G, and discriminators D_1 and D_2 correspondingly. The first line of Equation (1) represents the L1 loss, a regularization term meant to reduce the blurring in the generator output [56]. The second line stands for the losses of both discriminators classifying the real data, and the last two lines are their losses of classifying the generated data.

At test time:

The resulting trained generator produces the entire image continuously piece by piece, first left-to-right, and then top-to-bottom, in accordance with the requested semantic maps.[10, 11] Refer to the Figure 3.15 for the schematic explanation.

[10] Same as in the subsection 3.4.3, the very first conditions $\mathbf{C}_o$ are an exceptional case, since there is nothing to draw these conditions from. This can be resolved by either (i) using a top snippet from some real seabed image (non-generated) and left snippet as fully black (because the water column on the far-left is always black). Or (ii) using random noise snippets, or (iii) conditioning on both completely black snippets. (iii) will be very accurate for the water-column on the far-left, and no quite accurate for the upper condition snippet. For (ii) and (iii) the first few rows of pixels of the mission will be low-quality and will need to be removed.

[11] Reasoning for left-to-right top-to-bottom generation: physically, AUV flies underwater along the y axes, which is also the time axes, so conceptually it makes sense to fill the picture in row-by-row (top-to-bottom last). In real simulator you would also prefer this order as then human operator can begin scrolling the mission along the time-axes, before the full mission finished generating (like real data would be used). However, from the prototype implementation point of view there is not much difference at all. Moreover, the current implementation already works with an actual image to the right side of the sonar and a mirror reflection of the left side one (that is how images get stored in memory when collected by sonar). So technically R2D2-GAN already does left-to-right and right-to-left within a single implementation.

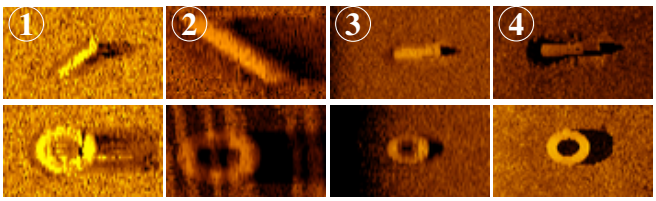

Fig. 3.16 **Examples of the two target objects we use for the ART tests (tyres and cylinders) and seabed types used for training the ATR:** 1. Uniform random noise background. 2. SonarSim-generated terrains[1]. 3. R2D2-generated terrains. 4. Real terrains. All of these targets are inserted into the terrains using the Cycle-GAN-based technique [123], because of the limited availability of the real data with real targets.

Generalisation:

The underlying technique of the R2D2 can be generalised beyond the specific GAN architecture. Nearly any GAN-based network, depending on the objectives and constraints of a specific task, can in principle be extended for the incremental recursive generation in a manner similar to the R2D2. The results provided in this subsection are based on extending pix2pix-style architecture [56] solely for the purpose of providing the user with full control over the topography of the synthesized mission via utilisation of semantic maps.

3.5.2 Experimental Setup

Visual quality tests:

A number of assessments were conducted in order to quantify the realism of the obtained imagery. We invited 10 domain experts to evaluate a selection of synthetic images created with the R2D2 and with classic pix2pix (for comparison), along with the real images. Participants inspected these images, labelling them as “real" or “synthetic" (“fake"). All the image sets of the compared methods were presented in even proportions. Furthermore, all of the image sets (both real and synthetic) correspond to exactly the same set of the semantic maps to ensure the best possible comparability. Images acquired from the different sources were shown sequentially, one at a time, in order to avoid the cognitive bias. For the same reason there was no prior information provided on the proportion of real vs. synthetic images. The only information provided was that the test set contains both real and synthetic images. The total amount of images assessed by a single expert during the visual assessment is 20 per method + 20 corresponding real images for comparison so 20×4, which results in 80 per person.

Although the time taken to inspect each particular image was recorded and analysed, there was no time constraints imposed on the participants during the test time.

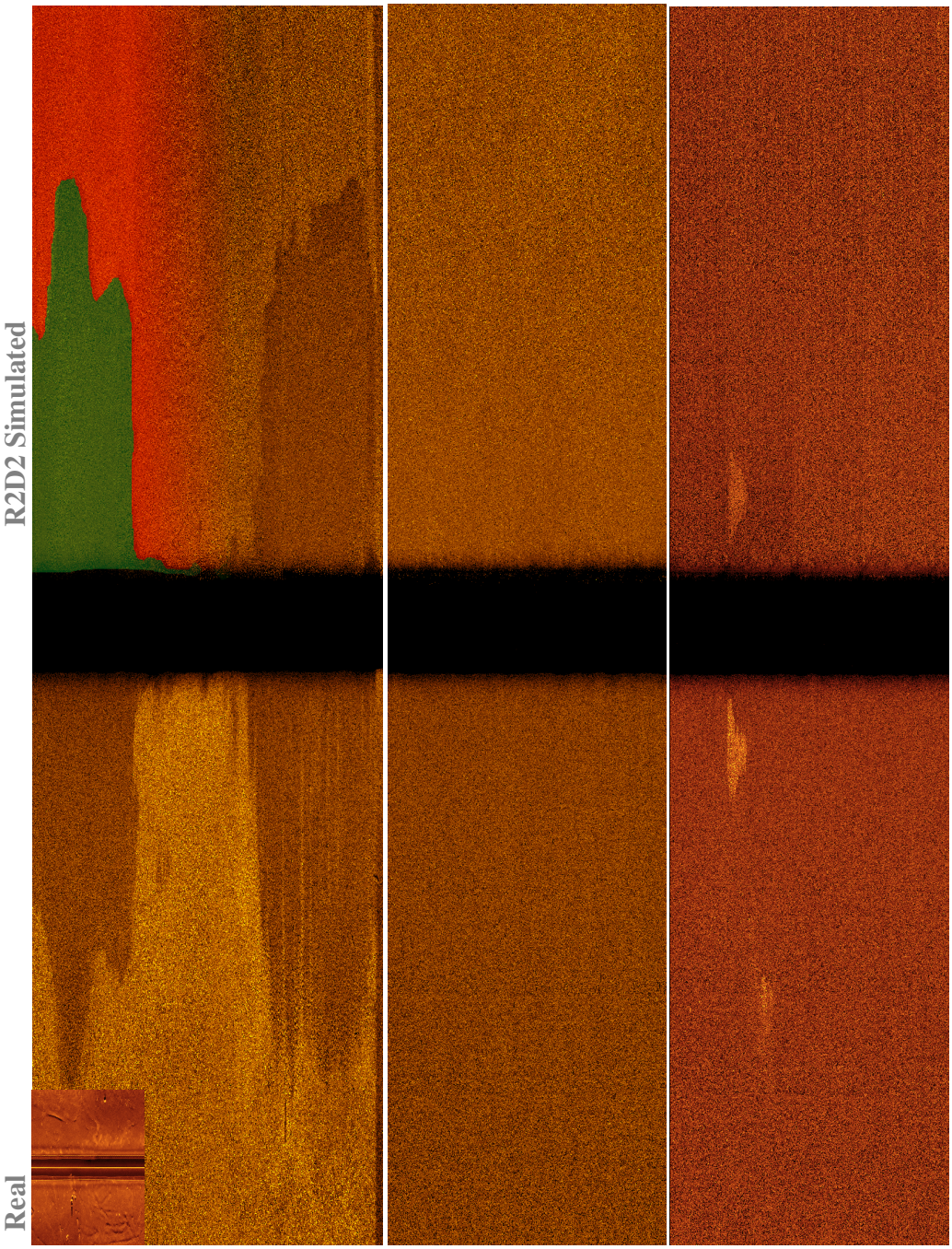

Fig. 3.17 **Visual results:** all the images have the real sonar scans on the left, and the R2D2-simulated images on the right. The horizontal pairs of images correspond to the same semantic maps. The miniature image at the top-left corner is the Marine Sonic sonar data, generated with the MC-pix2pix method [6]. The rest of the images are EdgeTech sonar scans, generated with R2D2-GANs, provided in a relative scale according to the corresponding across track resolutions. The partial overlay in the top-right corner is an example of the semantic map, used by a generator network in order to control the topography of the simulation according to preferences of the user.

Metrics:	fake labelled 'real'	labelling accuracy	average time
pix2pix	0.14 (±0.11)	0.88 (±0.29)	4.79 (±2.53)
R2D2 unnormalized	0.26 (±0.15)	0.82 (±0.17)	4.80 (±2.49)
R2D2	**0.78 (±0.25)**	**0.56 (±0.12)**	**5.23 (±3.43)**

Table 3.3 **Visual test results:** the experiment was conducted with participation of 10 human experts possessing the daily experience of dealing with the sonar imagery. They were shown an equal number of images generated by different sources (both simulated and real) and asked to label them as "real" or "fake". Images coming from different sources were shown one after the other in a random order to mitigate the cognitive bias. R2D2 images were labelled "real" in 78% cases, which compares well with the benchmarks, as well as with real images labelled "real" (90%). Humans also were able to distinguish it from real with accuracy of 56 %, which is close to random chance in a two-class problem ("real" / "fake"). The last column of the results shows how long (in seconds) on average it took to classify an image. R2D2-produced images took significantly longer to process, compared to the other methods.

Autonomous target recognition (ATR) training:

We argue that our proposed technique does not solely look good to human eye, but also can be of help for the autonomous systems training. For instance, assuming the lack of training data for the ATR, one could boost the training with the R2D2-generated data. Unfortunately, we do not possess unrestricted real data for sonars in EdgeTech resolution range (4620 or higher across track resolution), that would contain any real objects. Which is why we use the Cycle-GAN-based technique [123] to embed the artificial objects. In our specific case two of them - cylinders and tyres, see Figure 3.16. In fact, there is a very small amount of unrestricted real seabed scans available, so we have to use what little real data we have for the test set.

There are a few training sets: (i) uniform random noise background, (ii) SonarSim terrains - flat and rippled (respectively easier and harder for ATR to learn to operate on), and (iii) R2D2-generated terrains. Finally, we test the trained ATR performance on the small amount of the real sonar images we have available.

We use a simple RetinaNet-type architecture for ATR, Retina-18-FPN, as explained in Subsection 3.4.5, trained from scratch for this experiment. Both the training and the test sets are rather small, due to the unavailability of the real data. Note that we do not claim the state-of-art level of ATR results here, only the relative benefit of using the R2D2-generated data in the absence of the real training data.

3.5.3 Experiments and Results

The visual results of the R2D2-GANs are shown in Figure 3.17, there is very little to no difference between the real (left) and synthetic (right) images. Also, please note the relative difference in scales between the typical data generated with MC-pix2pix (top-left corner) and R2D2-GANs (right). Because of the iterative recursive generation along both axes, R2D2 is practically unlimited in the data resolution it is able to generate. Naturally, there is always a trade-off between the magnitude of the generated image resolution and the generation speed.

Image Assessment Scores

The image assessment scores by human experts, as presented in the Table 3.3, are based on the results collected from 10 human experts with various level of expertise, but dealing with sonar imagery on a daily basis. The total amount of images assessed by a single expert during the visual assessment is 20 per method + 20 corresponding real images for comparison so 20×4, which results in 80 per person. The individual assessment metrics are as follows:

(i) R2D2 synthetic imagery is labelled “real" by humans in 78% of the cases. This is the highest score across the competing methods, which also compares reasonably well to 90% score for the real images classified as real.

(ii) Human classification accuracy being close to 50%, i.e. near random chance for two-class problem (“real” / “fake”), indicated inability of humans to tell apart the real and synthetic images. The R2D2 shows the lowest human classification accuracy score of 56%. This is comparable with the 52% score obtained by the MC-pix2pix in an identical experiment [6], which is a remarkable result considering the significantly higher complexity of the current task of the higher-resolution generation.

(iii) We do not attribute any definite meaning to the average time spent on inspection of each separate image. Nevertheless, images produced by R2D2-GANs take the longest to classify. We suggest to interpret this as the R2D2-generated synthetic imagery posing the higher challenge for distinguishing it from real.

Our method outperforms the original pix2pix according to all the metrics in this assessment. It is also comparable with the current state of art - MC-pix2pix [6], which achieves the human labelling accuracy of 52% for images of much smaller resolutions. The R2D2, however, surpasses this competitor by the resulting resolution of the complete images it is capable of generating, while maintaining the comparable quality of the generated results.

Train set:	Noise	SonarSim (flat)	SonarSim (rippled)	R2D2-GAN
Recall	0.00	0.3314	0.2255	**0.4843**
F1	0.00	0.4895	0.3653	**0.6073**

Table 3.4 **Autonomous Target Recognition experimental results:** unfortunately, we have no access to unrestricted data with real targets at our disposal. However, we demonstrate, that expanding the ATR training datasets with R2D2-generated data may help the learning process. Potentially, a higher variety of the terrains available at training time should help the ATR system to generalise better. Fortunately, there is a Cycle-GAN-based method [123] to insert some artificial objects into the terrains, that is useful in this case. The training is conducted on 4 different types of terrains: uniform random noise, SonarSim[1] flat and rippled terrains (respectively less and more challenging for the ATR), and the R2D2-generated terrains. We train a simple ResNet-type network over these, and test on the real data with artificial targets embedded.
The results suggest that random noise in this case is completely useless, whereas R2D2-GAN on the contrary performs better than the competitors.

Generation speed

R2D2 was developed for the larger resolution sonars, such as EdgeTech. EdgeTech has the speed of acquisition of 160 000 pixels per second. R2D2 has the speed of generation of about 2.2 times faster than that (again, tested on GTX 1080 Ti (12GB RAM), implemented in Pytorch, including image loading/processing time.

ATR performance results

ATR perfromance results are available in Table 3.4. Both recall and F1-score[12] suggest that the R2D2-generated terrains provide significantly better training material compared to the random noise and SonarSim simulator.

3.6 Conclusion

This chapter presented MC-pix2pix and R2D2-GANs - novel techniques for generating realistic synthetic imagery of any specified resolution and topography via sequential conditional generation. We provided both the quantitative and qualitative evidence confirming the realism of the images produced with these methods. The empirical assessments also suggest significant advantages for the ATR systems trained with synthetic data generated by MC-pix2pix and R2D2.

[12]F1-score - harmonic mean between the precision and the recall of the ATR system. Higher values correspond to the better performance.

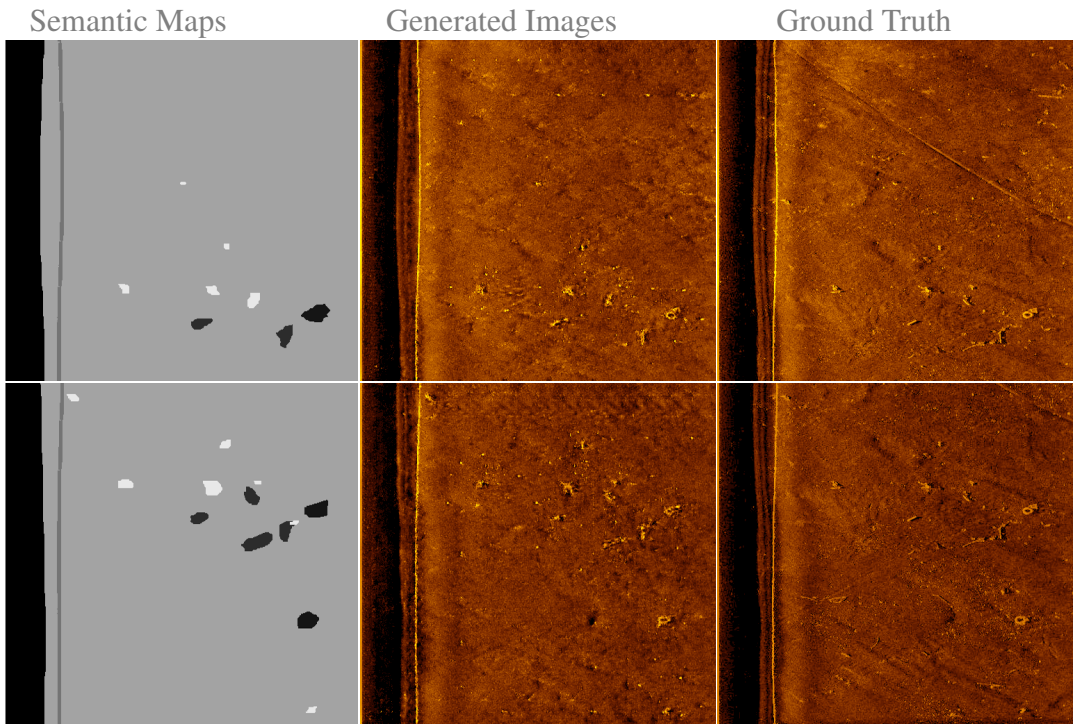

Fig. 3.18 **Objects and artifacts:** only a very limited amount of data containing artefacts and objects was available to us. These visual results are coming from MC-pix2pix trained on 100 training images only. Semantic maps on the left are grey-scale here, that is what they look like without yaw data overlay. Different shades represent different classes - terrains, visual effects, and objects - two darkest shades stand for tyres and cylinders. Although the generated images demonstrate object and artefact generation in principle, a more thorough investigation with ideally more data and more variety in types of artefacts and object could be a topic for the whole new research project. That is if the suitable data ever become available.

The presented techniques are in principle compatible with nearly any type of GANs, which might be of benefit for alternative objectives than those explored in this work. Thus providing the user with the ultimate control over the exact nature of the preferred data generation process.

The R2D2-GANs are practically unlimited in the image resolution they can generate (at expense of the generation speed), which makes them immediately applicable to even higher resolution sonars, such as the Synthetic Aperture Sonars. Nonetheless, in the future work we intend to optimise this method further to make sure the fastest possible speed of the image generation for even higher resolutions.

3.7 Potential Directions for the Future Research

Dealing with Artefacts. Sonar data often feature various visual artefacts, such as surface return for instance - a bright vertical line, running in parallel to the water column, caused by the reflection of the sound from surface when the vehicle is relatively close to the surface. These are usually of no interest for practical reasons, however if needed they can be easily reproduced as well by labelling the surface return as a separate class at train time, and then using the pseudo-altitude data for reasonably placing this label into the semantic maps to be translated into a realistic output. Some examples of the results are available in Figure 3.18.

Dealing with Objects. We have used [123] for inpainting objects into the simulated images solely because we did not have an access to data containing sufficient amount of the training examples of objects. It is however a trivial extension to produce any type of desired object within MC-pix2pix and R2D2 pipelines just by adding different types of objects as additional classes. Figure 3.18 has some examples of this, they are of limited quality because of the extremely small size of the available dataset containing the objects.

Potential use of MC-pix2pix for semantic segmentation. The translation direction could be reversed from realistic images to semantic maps instead, in order to create a segmentation algorithm. It is a well-researched problem for multiple types of image data from aerial views segmentation [127] to medical image segmentation [128]. Some visual results of the most basic version of MC-pix2pix-based segmentation applied to sonar image segmentation are provided in Figure 3.19. To give you a rough idea, the accuracy we instantly achieved with the minimal post-processing of the resulting segmented images is $93.63\% \pm 7.85\%$ on average across 9 texture classes. However, an in-depth investigation is needed to

(i) figure out where the artefacts from Figure 3.19 (bottom row, middle image) come from,

(ii) what are the accuracy scores within the texture classes for the class-balanced training datasets,

(iii) how shifting the class balance within a training dataset might affect the performance,

(iv) what is the influence of adding more classes.

It is an interesting and useful direction, however, since it is enough work for the whole new research project, we leave this further investigation for the future work.

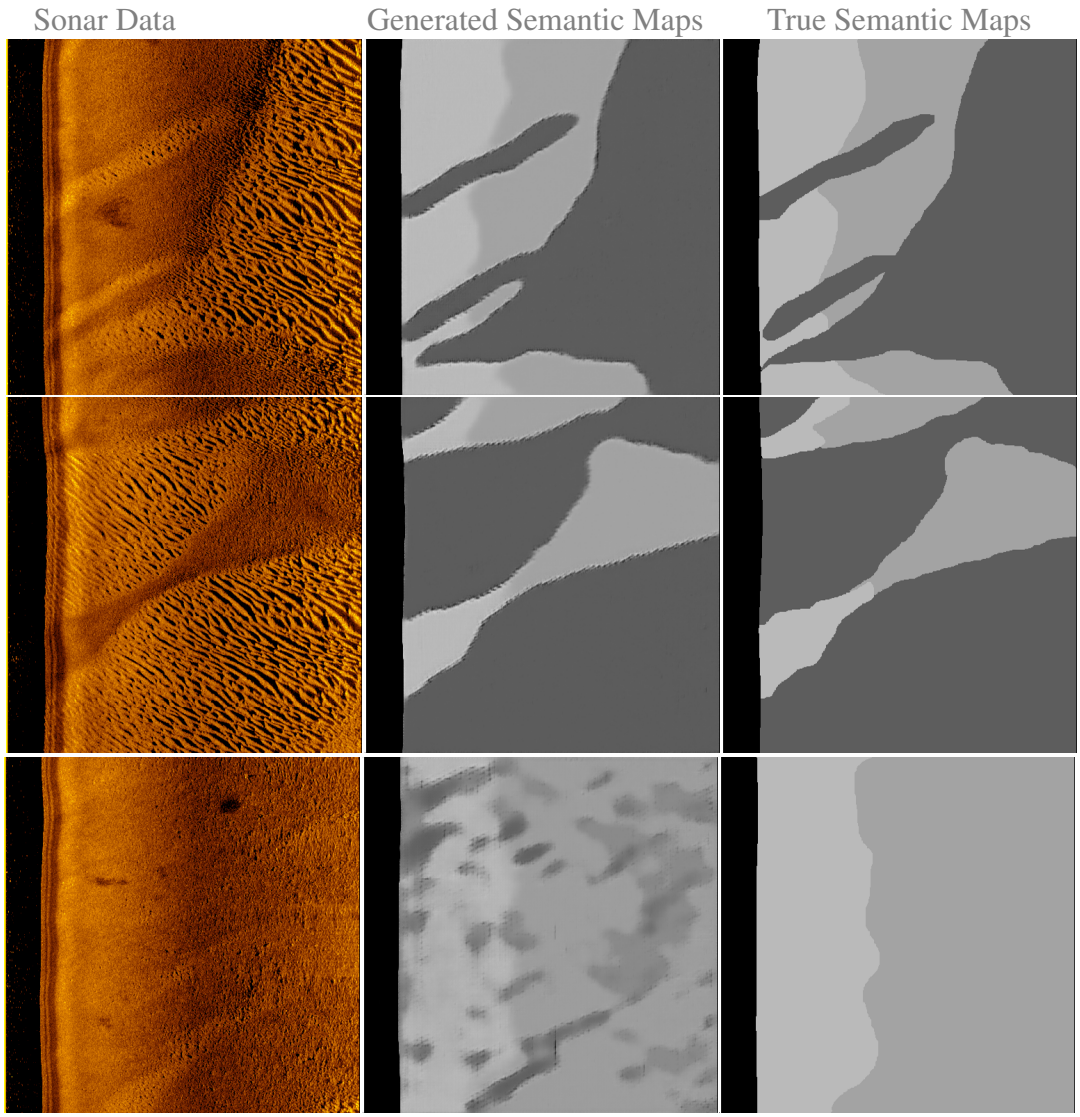

Fig. 3.19 **Semantic Segmentation with MC-pix2pix-style network:** this application would pretty much require to just reverse the previously explained order of things - instead of translating from semantic maps into realistic sonar scans, we would train MC-pix2pix to translate from realistic looking images into the semantic maps. The resulting segmentation is mostly pretty accurate, other than the blurry borders between the classes (please compare the middle column to the right column). However, it occasionally returns weird artifacts, that look like the ones in the last example (bottom row middle vs. right). This might be down to the imprecise manual labelling used to acquire the semantic maps used for training the segmentation network, however verifying this guess would require considerable additional investigation.

Chapter 4

Generative Policy Networks for Behavioural Repertoires Generation[1]

Learning algorithms are enabling robots to solve increasingly challenging real-world tasks. These approaches often rely on demonstrations and reproduce the behavior shown. Unexpected changes in the environment or in robot morphology may require using different behaviors to achieve the same effect, for instance to reach and grasp an object in changing clutter. An emerging paradigm addressing this robustness issue is to learn a diverse set of successful behaviors for a given task, from which a robot can select the most suitable policy when faced with a new environment. In this paper, we explore a novel realization of this vision by learning a generative model over policies. Rather than learning a single policy, or a small fixed repertoire, our generative model for policies compactly encodes an unbounded number of policies and allows novel controller variants to be sampled. Leveraging our generative policy network, a robot can sample novel behaviors until it finds one that works for a new scenario. We demonstrate this idea with an application of robust ball-throwing in the presence of obstacles, as well as joint-damage-robust throwing. We show that this approach achieves a greater diversity of behaviors than an existing evolutionary approach, while maintaining good efficacy of sampled behaviors, allowing a Baxter robot to hit targets more often when ball throwing in the presence of varying obstacles or joint impediments.

[1]This chapter consists of the materials published as a conference paper 'Behavioral Repertoire via Generative Adversarial Policy Networks' in IEEE International Conference of Developmental Learning and Epigenetic Robotics 2019, as well as its journal extension accepted for publication in the special issue of IEEE Transactions on Cognitive and Developmental Systems.

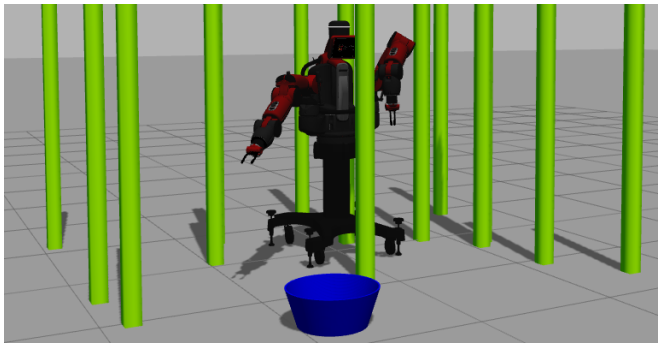

Fig. 4.1 An example obstacle-occluded environment, for which the GPN-generated policy repertoire can offer solutions.

4.1 Introduction

Robots are increasingly able to solve challenging tasks by learning controllers. While reinforcement or imitation learning approaches can be effective, they typically learn a single ideal solution to a given control problem, and the robustness of that solution to challenging situational variants (e.g., changing/complex obstacles, such as in Fig. 4.1, or damage to the robot) is hard to guarantee. If a control policy fails due such an unexpected environmental change, robots can try to adapt their control policy to a new situation through re-planning [129] or adapting a learned policy [130, 131]. Beyond such adaptation, when animals face a challenging environment in which a previously learned behavior fails, they also draw on an additional capability: leveraging a suite of other known behaviors that are expected to solve the task at hand [132]. Exploration within a set of diverse historical behaviors that solved a task can quickly lead to a solution that succeeds in a new environment [132]. Such behavioral repertoire-based approaches are emerging as promising techniques for robustly solving tasks [132–134].

Existing realizations of this robustness-through-diversity vision are often based on evolutionary algorithms that train a diverse set (population) of controllers that solve a given task [133, 134]. However this approach has several drawbacks: storing a large database of controllers is not compact, and there is only as much diversity as is contained in the population of controllers. We argue that a preferable instantiation of this vision is to learn a generative model over controllers. Firstly, it is compact – only the parameters of the generative model rather than a large list of controllers need to be stored. Secondly, the available diversity is not limited to the instances in a fixed length list. By sampling a generative model over controllers, an unlimited number of distinct controllers can be obtained. And with a sufficiently flexible generative model, sampled controllers need not be simple interpolations between controllers used to train the generative model.

Samples could encode novel solutions to the problem by drawing diverse aspects of multiple training policies.

This approach is coherent with the exploratory behavior of infants (and other animals) – specifically their ability to perform a behavior in high variation so there is a distribution of actions associated with each behavior [135]. Our method models this distribution. Following the example of other progressive sequential architectures – [136, 137], we propose a simple two-staged developmental framework where one first builds up the initial repertoire of actions (using methods such as quality-diversity search [138]), and then generalizes beyond this repertoire via our proposed generative model. Our progression from a library-based approach to a generative-model can also be considered a representational re-description [139], between developmental waves [135].

While conceptually appealing, training generative models over policies is non-trivial. The space of reasonable policies likely to solve a given task is a complicated manifold within the space of all policies, considering actuator redundancy, non-linearities and so on. We therefore propose to apply generative adversarial networks (GANs) [140] to model the distribution over policies that solve a given task using a neural network, thus defining a generative policy network (GPN). In our framework the GPN models the distribution over policy parameters, so that each sample from the GPN defines a specific robot controller. Multiple samples from the GPN therefore correspond to different solutions to the task that the GPN is trained on. To generate training data for the GPN we exploit quality-diversity (QD) search evolutionary algorithms [138] to find a diverse set of policies that solve a task. Once trained, a GPN then provides a compact source of diverse and novel policies likely to solve variants of that task. Compared to a conventional GAN, we find it beneficial to regularize GPN-training by requiring it to generate not only a controller but the outcome of running that controller (i.e. to simulate the forward model, or reconstruct the input goal state), and this is also useful as a way to pick promising policies (e.g., sample the GPN until a policy is drawn which is expected to work in the current environment).

We demonstrate our approach through the specific application of target-conditional ball-throwing [141, 142] in the presence of confounding obstacles or joints impediments. Throwing is often formalized as a contextual policy problem where a movement primitive for throwing is synthesized conditionally on the desired target position [141, 142]. In the presence of obstacles however, the most 'natural' way to throw to a given target may be blocked. Nevertheless, there are multiple throwing movements that hit a given target. We show that the ability to model – and sample from a distribution of controllers

allows the robot to find throwing controllers that can avoid any given motion constraint. A preliminary version of this work appeared in [5].

4.2 Related Work

Learning Robot Control Typical approaches to learning robot control include learning by demonstration [143], reinforcement learning to maximize some extrinsic reward [141], or demonstration-based initialization followed by policy search-based reinforcement learning to fine-tune the demonstrated policy. Those policy search algorithms in turn can often be categorized into gradient-based [144, 145] and gradient-free methods such as Bayesian optimization [146] and evolutionary search [147]. An advantage of evolutionary methods is that they often provide a population of policies as a byproduct, rather than a single best controller.

Where obstacles can impede behavior, a standard robotics approach is to localise the obstacle and plan a movement that avoids it [148]. However this requires both (i) accurate 3D obstacle localisation and (ii) appropriate adaptive planning capabilities. One or other of these sensing and reasoning capabilities may not be available at the required efficacy level at a given developmental stage in an animal or robot. In contrast generating diverse behaviors and exploring them until one works has lower prerequisites and hence is suitable for earlier developmental stages.

Behavioral Diversity in Robot Control For a robot to be able to deal rapidly with new and unanticipated situations, a recently proposed approach consists of building a large repertoire of behaviors in which it should be possible to find one adapted to a newly arising situation or environment. The repertoire creation step can be done in a preliminary phase and a learned repertoire subsequently used to accelerate the adaptation to an unanticipated situation by relying on a selection process instead of a full learning process [132]. Promoting behavioral diversity is a key feature of a repertoire creation process. Driven by research on novelty search [149], evolutionary approaches have been adapted to generate a behaviorally diverse set of solutions instead of converging to a single solution optimizing a given fitness function [133]. These algorithms are called Quality Diversity algorithms [150, 151] and are used here to bootstrap the proposed method. Our proposed GPN builds on QD-search by leveraging its results as training data. However, in contrast to the selection-from-repertoire paradigm of QD, it has several interrelated benefits: (i) We can more compactly store a large repertoire by storing instead the parameters of a generative model that represents that repertoire of

behaviors[2]. (ii) Rather than a fixed size database of behaviors, the generative model can continue to sample unlimited new behaviors until a suitable one is found. (iii) Samples drawn from the generative model of behaviors can discover novelty beyond the initial training repertoire, by combining aspects from different training behaviors. (iv) Importantly the GPN approach is better suited for contextual policies. To solve a contextual policy task like diverse throwing to different targets, a library-based approach increases the required data collection and repertoire storage size dramatically because it would need to keep samples of many different throwing targets, and for each of those targets, samples of many different ways to throw there. In contrast, given a few samples of different throwing targets, a contextual policy GPN can extrapolate and draw many different controllers for throwing to any given target.

Generating Diverse Policies Another somewhat related work [152] covers diverse policy generation in a model-based framework. DIAYN [152] learns diverse skills (policies), assessing the diversity by the variety of states they visit in the process of RL-style unsupervised exploration. The main difference is that DIYAN tries to learn a small set of very distinct skills. While our GPN focuses on one skill type, but learns an infinite smoothly-varying manifold of controllers covering both all the potential goals (e.g., movement targets) and ways to achieve those goals. DIYAN also focuses more on initial exploration (and is thus analogous to QD-search in our pipeline), while we focus on compactly representing and exploiting the results of such an exploration process.

Generative Adversarial Networks Generative Adversarial Networks were proposed [140] to address the challenge of learning a neural network-based generative model for complex high-dimensional data. The key idea being that generator training is enabled by a second discriminator network that is simultaneously adversarially trained to distinguish true training data and the generator's synthetic examples. To improve its ability to fool the discriminator the generator must generate increasingly realistic synthetic samples. There have since been numerous extensions including convolutional GANs [153], conditional GANs [154, 155], disentanglement and interpretable latent codes [156], and improvements of GAN training stability with regards to challenges such as non-convergence and mode-collapse [**?** 65].

Generative Adversarial Network Applications The vast majority of GAN applications are in image generation tasks [140, 153, 157**?** , 155]. In robotics, GANs have been applied in robot haptic recognition [158]. Autoencoding VAE-GAN has been used for

[2]Compact in principle - depending on the number and kind of the controllers needed. QD, as a search algorithm, takes time to produce valid trajectories, moreover it does so at random, compared to the GPN operating in a target-conditional manner. Further in this chapter a trained GPN generator is roughly comparable in size with the initial QD library that GPN was trained on, which makes GPN a better choice in principle if we want to expand beyond size of the original QD training data set.

visual representation learning to process visual input in support of vision-based actuation in control [159]. The most related application of GANs has been in an imitation or inverse reinforcement learning context by GAIL and InfoGAIL [160, 64]. GAIL trains a *single policy* by imitation learning by matching generated and demonstrated state-action pairs distribution. In contrast, GPN trains a *distribution over policies* by matching generated and demonstrated policy distribution. Furthermore, all GPN distribution generation is target-conditional, while GAIL has no conditioning. InfoGAIL extends GAIL to a multiple expert setting, training a *small discrete set of policies* by matching a generated policy distribution conditioned on a discrete latent variable, and a demonstrated set of policies. In contrast GPN trains a *continuous distribution over policies* by distribution matching. GPN enables conditioning on a continuous variable such as target, while InfoGAIL conditions by selecting a discrete policy. To our knowledge neural network generators have not previously been applied to sample diverse continuously distributed control policies, or to the generation of diverse robot behaviors, as we explore here.

4.3 Method

4.3.1 Generative Policy Networks

Unconditional Policies Robot behaviors are defined by a control policy π operating in some state space $\mathscr{S}$ and action space $\mathscr{A}$. Thus while generative models are conventionally used to define a distribution $p(\mathbf{x})$ over data instances $\mathbf{x}$, our GPN defines a distribution $p(\pi)$ over policies π, which are themselves functions $\pi : \mathscr{S} \rightarrow \mathscr{A}$. Given a set of training policies $D_{train} = \{\pi_i\}$, we train our GPN to estimate the distribution over observed policies.

Assuming the policies in question lie in some parametric family, then each is identified by some parameter vector (e.g., weights in a neural network [131], radial basis function (RBF) kernels in a dynamic movement primitive (DMP) [143, 142]). By training a generator G_ϕ to generate such parameters, samples from the generator are interpretable as controllers. In this case the discriminator enables the training of the generator by learning to distinguish between real policies in D_{train} and generator synthesized policies $\pi = G_\phi(\mathbf{z})$. Once the generator learns to fool the discriminator, and assuming it does not mode collapse, then samples from the generator represent diverse control policies that are novel yet statistically indistinguishable from the training policies. We denote sampling policies from the distribution implied by the generator $G_\phi(\mathbf{z})$ under a given noise distribution $p(\mathbf{z})$ as $\pi \sim p_{G_\phi}(\pi)$.

Contextual Policies We aim to go beyond simple fixed behaviors to work with contextual policies that are parameterized by a goal condition to achieve [141, 143]. For a goal directed policy such as our intended application of robot throwing, we need not just a controller (e.g., a throwing movement), but a conditioning mechanism [154, 156] that generates a controller that achieves the right goal (e.g., a throwing movement that hits a specific target). As described above, if the training set D_{train} consists of controllers throwing to multiple different locations, then sampled policies $\pi \sim p_{G_\phi}(\pi)$ will throw to new locations within the distribution of training targets. If the training set D_{train} consists of multiple controllers that throw in different ways to the same location, then $\pi \sim p_{G_\phi}(\pi)$ will sample novel policies that throw to that same location. In the contextual policy case we want a policy that achieves a specifiable goal. Thus we define a conditional generator $\pi = G_\phi(\mathbf{z}, \mathbf{c})$, $\pi \sim p_{G_\phi(\mathbf{c})}(\pi)$ to sample policies π that target a specific landing point $\mathbf{c}$. Thus we can both throw at a specified target (set the generator condition), and also find multiple ways to throw there (sample the generator).

4.3.2 Application to Throwing

For application to throwing, we assume a set of training policies, and denote sampling these as $\pi \sim p_{data}(\pi)$ and $\pi, \mathbf{c} \sim p_{data}(\pi, \mathbf{c})$. We then train a conditional generator network $G_\phi(\mathbf{z}, \mathbf{c})$ as below. The third term is an added regularizer that requires the generator to reconstruct the landing point of the policy that it just sampled. Here $G_\phi(\mathbf{z}, \mathbf{c})_T$ means sample the policy and its target point and take only the target point term, and $G_\phi(\mathbf{z}, \mathbf{c})_{\neg T}$ means the opposite.

$$\begin{aligned} \min_{G_\phi} \max_{D_\psi} V(G_\phi, D_\psi) = & E_{\pi, \mathbf{c} \sim p_{data}(\pi, \mathbf{c})}[\log D_\psi(\pi, \mathbf{c})] \\ & + E_{\mathbf{z} \sim p(\mathbf{z}), \mathbf{c} \sim p_{data}(\mathbf{c})}[\log(1 - D_\psi(G_\phi(\mathbf{z}, \mathbf{c})_{\neg T}))] \\ & + \left\| G_\phi(\mathbf{z}, \mathbf{c})_T - \mathbf{c} \right\|_2 \end{aligned} \tag{4.1}$$

Policy Representation We have applied our framework successfully to many different policy representations including sampling the RBF parameters of the forcing term of a DMP, but we found the following simple representation effective and easy to tune. For our Baxter robot arm, we represent the π as a 15D vector defining a high-level open-loop controller in terms of the ball release time, and effector position and velocity at release. Specifically $\pi = [\boldsymbol{\theta}_{t_T}, \dot{\boldsymbol{\theta}}_{t_T}, t_T]$, where $\boldsymbol{\theta}_{t_T}$ and $\dot{\boldsymbol{\theta}}_{t_T}$ are 7D robot arm joint angles and joint velocities at launch time, and t_T is the launch time.

The goal condition **c** is a 2-dimensional Cartesian coordinate of the ball landing point. There are multiple launch configurations as described above, that result in the same landing point **c**, and the trained GPN will sample this space of configurations.

Policy Execution With the policy definition above, samples from our GPN constitute a high-level action plan of how to launch the ball. To actually actuate this we map the high-level action into an open loop controller for low-level actuation via the following third-order polynomial function of time:

$$\boldsymbol{\theta}_{t_i} = \alpha_4 \left(\frac{t_i}{t_T}\right)^3 + \alpha_3 \left(\frac{t_i}{t_T}\right)^2 + \alpha_2 \left(\frac{t_i}{t_T}\right) + \alpha_1 \tag{4.2}$$

with $\dot{\boldsymbol{\theta}}_{t_i} = \frac{d\boldsymbol{\theta}_{t_i}}{dt_i}$ and $\ddot{\boldsymbol{\theta}}_{t_i} = \frac{d\boldsymbol{\theta}_{t_i}}{dt_i}$, and parameters:

$$v_{t_i} = \frac{1}{t_T}\left(3\alpha_4 \left(\frac{t_i}{t_T}\right)^2 + 2\alpha_3 \left(\frac{t_i}{t_T}\right) + \alpha_2\right), \; \alpha_1 = \boldsymbol{\theta}_{t_0},$$
$$\alpha_2 = \dot{\boldsymbol{\theta}}_{t_0} t_T, \quad \alpha_3 = 3\boldsymbol{\theta}_{t_T} - \dot{\boldsymbol{\theta}}_{t_T} t_T - 2\alpha_2 - 3\alpha_1,$$
$$\alpha_4 = \boldsymbol{\theta}_{t_T} - \alpha_1 - \alpha_2 - \alpha_3,$$

where $\boldsymbol{\theta}_{t_i}, \dot{\boldsymbol{\theta}}_{t_i}, \ddot{\boldsymbol{\theta}}_{t_i}$ are the positions, velocities and accelerations of joints at time t; $\boldsymbol{\theta}_{t_0}$ and $\dot{\boldsymbol{\theta}}_{t_0}$ are initial positions and velocities at time t_0 and t_T is the time of launch. We assume the starting robot arm configuration is the same for each trial. Baxter is then actuated by sending the above joint position, velocity and acceleration plan to a ROS control node.

4.3.3 Data Collection with QD Search

We describe the collection of data used to train our GPN. Please note that, our main contribution of GPN is agnostic to the specific method used to collect training data. In our experiments we use data collected by archive-based Quality Diversity (QD) search [161], following the principles of Novelty Search with Local Competition (NSLC) [138], throughout, although other data sources could be used. Specifically, we use the evolutionary QD search method [138] to find a set of genotypes (high level policies π) that have diverse behavior-space effects (e.g., arm trajectories and landing positions) when actuated, while being of high quality (low torque during the entire actuation). Specifically behaviour-space effect is measured by describing each motion in terms of an 11D phenotype feature consisting of landing point of the ball, and 3 sets of 3D arm position descriptors recorded during the throwing. Diversity is then measured via the negative exponential distance between a behaviour and the dataset so far. QD search then optimizes for a high quality an diverse dataset. Please see [162] for full details on QD data generation for throwing. Overall, we use use a realistic Baxter simulation

(Gazebo) to obtain around 15,000 throwing episodes. Each training episode records the arm trajectory, ball trajectory, and ball landing point. Since landing points are continuous variables, there is only one trajectory per landing point. Each arm trajectory is defined by a single $15D$ set of parameters, all of the trowing trajectories start in the same initial position for simplicity sake.

4.3.4 Baselines for Comparison

Evolutionary Repertoire Baseline We use QD search to generate training data for our GPN as described above, and will exploit our trained GPN to solve a robust throwing task later. For quantitative comparison, we consider an alternative evolutionary-style approach to exploiting this data. Such a repertoire-based approach to robust throwing would treat the dataset as a large library (repertoire), and then solve a new task by selection from the repertoire [132–134]. To throw to a specific target, the closest memorized landing point is recalled, and the associated policy is executed. If an environmental change (e.g., an obstacle) causes that known solution to fail, a lookup can be performed to find and execute some other policy with approximately the same landing point, but potentially different arm/ball trajectory. The problem is that this scales badly: although a large number of throwing episodes (15,000) covers the space of landing points reasonably well, it is not enough to cover many diverse ways to throw to each individual landing point. So the lookup-based approach may fail to effectively find diverse ways to throw to a specific point.

Other Alternatives Besides QD, we also compare two general purpose alternatives to GPN for robust throwing. KDE: As a non-parametric alternative to our GPN, we define a target-conditional Kernel Density Estimation [47] model over the same QD-based training set used by our GPN, so $p(\pi|\mathbf{c})$ is a Gaussian mixture model. We can then sample this mixture instead of our GPN. BayesOpt: Bayesian Optimization [163] is an established approach to adaptive behaviors in robotics [146]. We use the the best QD-trajectory for the given targets as the starting condition, and then perform Bayesian optimization for 10 trials for direct comparison to the diversity-based models.

4.4 Experiments

4.4.1 Training data and settings

We apply our GPN to enable a Baxter robot to robustly throw a ball in different environmental obstacle conditions, as well as within various joint impediment scenarios.

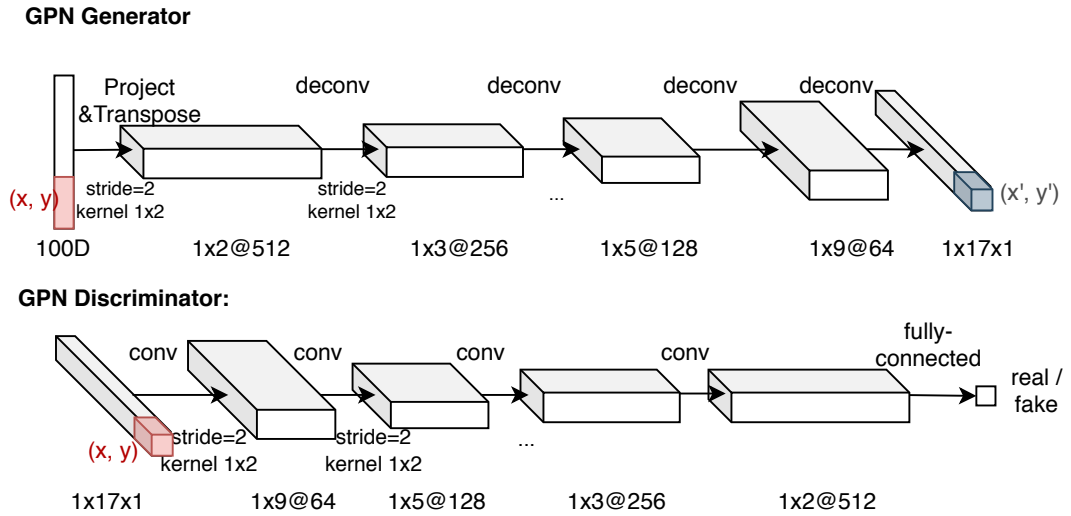

Fig. 4.2 **GPN Generator:** inputs noise vector concatenated with the target coordinates (x,y), gets it through one fully connected and 4 deconvolutional layers (all: strides 2, padding 1, kernel size 1×2, ReLu-activated, except for the last layer that has Tanh activation). The result is 17D output where 15D is trajectory parameters and the remaining two are predicted landing point of the trajectory (x',y') - we penalise generator on ability to output that correctly and show the difference in accuracy it makes in Figure 4.5.
GPN Discriminator: receives the 17D trajectory parameters and landing point (trajectory parameters can be either real or generated, but the landing point is always real, as the condition must be). Takes it through mirror reflection of the generator - 4 convolutional layers (all: strides 2, padding 1, kernel size 1×2, LeakyReLu-activated, except for the last layer that is fully connected with sigmoid activation. The output is the prediction of whether the discriminator thinks identifies policy as real or fake.
For the rest of the details on training target-conditional GPN please refer to sections 4.3.2 and 4.4.1.

Since the Baxter arm has 7 joints, there are 15 parameters for any policy (Eq. 4.2) – position and velocity for each joint and the launch time. The training data is illustrated in Figure 4.3 in terms of a heat map of ball landings at different positions on the floor around the robot (left), and some example training episodes represented by their arm trajectories, ball trajectories and landing points (right).

Settings Our GPN is built upon DCGAN framework [153], and has a 4-layer RELU-activated convolutional architecture that maps a 100-dimensional noise vector $\mathbf{z}$ to a 15-dimensional output vector representing π. It is trained using 15000 data instances for 1000 epochs, with batch size 250, using Adam optimiser with learning rate 0.0002 and $\beta_1 = 0.5$. We used 20 generator updates for each discriminator update. Both QD and GPN use the same policy representation π, and underlying actuation strategy. Please refer to Figure 4.2 for the rest of the details on the target-conditional GPN architecture.

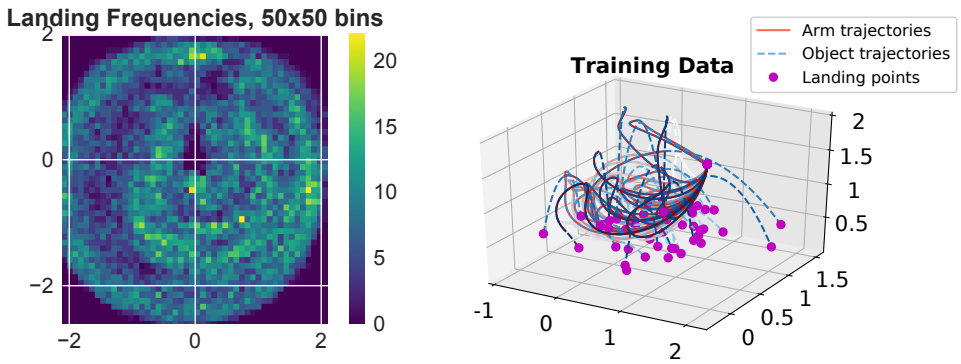

Fig. 4.3 Training data for throwing. Baxter robot is located at $(0,0)$. Left: Frequency of ball landing points at different floor positions (50×50 bins). Right: Example trajectories where red line is the end-effector, blue lines are the ball in flight, and magenta points are the landing points on the floor.

Evaluation Metrics We evaluate the methods with two sets of metrics. For evaluating simple throwing, we compute: **RMSE** between the target and actual landing point for all the trials; **Diversity** of the trials by taking the ball trajectory, computing equidistant waypoints along it, and then using these to compute a standard deviation of all trajectories towards a given target; **Harmonic Mean** aggregates the other two metrics - accuracy $(1 - RMSE)$ and diversity (standard deviation) in order to provide a single quantitative measure of performance. Note that harmonic mean aggressively penalizes failure in either metric.

When evaluating the ability of repertoires to perform robust throwing in the presence of obstacles, or with broken joints, we consider a trial as a success if the ball lands within radius τ of the intended target. Our metric is $SuccessesProportion(k,\tau)$: How many of the target coordinates does the ball hit successfully (within τ radius), at least k out of 10 times, when sampling from the repertoire? The idea is that even if obstacles block some particular throws, or a physical malfunction causes it to fail, a model that can generate multiple diverse behaviors that all solve the task (i.e., throwing trajectories that hit the same target) should be able to find at least some (i.e., k) successful solutions.

4.4.2 Experiment 1: Target-conditional throwing

Setup We aim to achieve robust throwing by learning to hit a target in diverse ways. We therefore first evaluate the ability of our GPN and QD alternative to: (i) accurately throw to a given position, and simultaneously (ii) find diverse ways of throwing to each position around the robot. For this purpose we grid the floor space around the robot into a 5×5 grid (25 target landing points). We experiment both in simulation (Gazebo Baxter) where we attempt to throw to each of those points 10 times, and then corroborate those results on the real Baxter where we throw to each coordinate 3 times,

tracking the ball using an OptiTrack system. We compare results from our GPN with the standard evolutionary strategy that treats the GPN-training set as a repertoire library (Sec. 4.4.1).

Results The results in Figure 4.4 plot the landing error and diversity metrics at each grid point on the floor around the robot. The first two columns compare QD/Library-based approach with our GPN in simulation; the third column evaluates our GPN results on the real Baxter robot. From the results of this experiment we make the following observations: (i) In general QD search has higher accuracy. This is expected as it is simply recalling previously memorized movements and replaying them exactly, which unsurprisingly leads to very similar outcomes, and hence high accuracy. In contrast our GPN is a predictive model that must infer the right policy to throw to any given target point, so its slightly lower accuracy is understandable. (ii) However, GPN has much higher diversity. It models the distribution of policies that throw to a conditioning target, and samples that distribution for each trial. (iii) Aggregating these metrics via harmonic mean, we see that our GPN performs favorably compared to QD. (iv) When executing our GPN-sampled controllers on the physical Baxter robot, the results are comparable to the simulated case (third vs second column). Note that the grey areas to the left of the map on the results of the real robot are because of walls in the physical Baxter environment preventing the data collection in these regions.

These results are summarized over all the spatial coordinates in Figure 4.5 (top), where GPN significantly outperforms QD in terms of harmonic mean. To be as fair as possible to the QD search alternative, we also considered boosting its diversity by adding Gaussian noise to the executed policies at each trial. The result in Figure 4.5 (bottom) shows that noise can improve QD performance. However it must be carefully tuned as too much noise quickly degrades QD's accuracy. Overall this result is understandable as uniformly adding noise to known throwing behaviors can quickly move off the manifold of good throwing policies. In contrast GPN learns the distribution over good throwing policies so it can sample novel throwing controllers from within that distribution. Finally we mention that, as per common safety practice, all our movement plans are checked for self-collision before execution. We also note that despite lacking a model of robot kinematics, the vast majority (98.4%) of the diverse plans generated by GPN are collision free. This indicates that GPN has also learned about the manifold of reasonable controllers in the sense of non-colliding as well as ability to hit a target.

[2]Symbolic representations of significance levels in this paper are as follows: '+' stands for the method or parameter against which the rest are compared (usually GPN for methods or the parameter we are using for the parameter sensitivity assessment), '*' stands for significance level $\alpha = 0.05$, and '**' - for $\alpha = 0.01$.

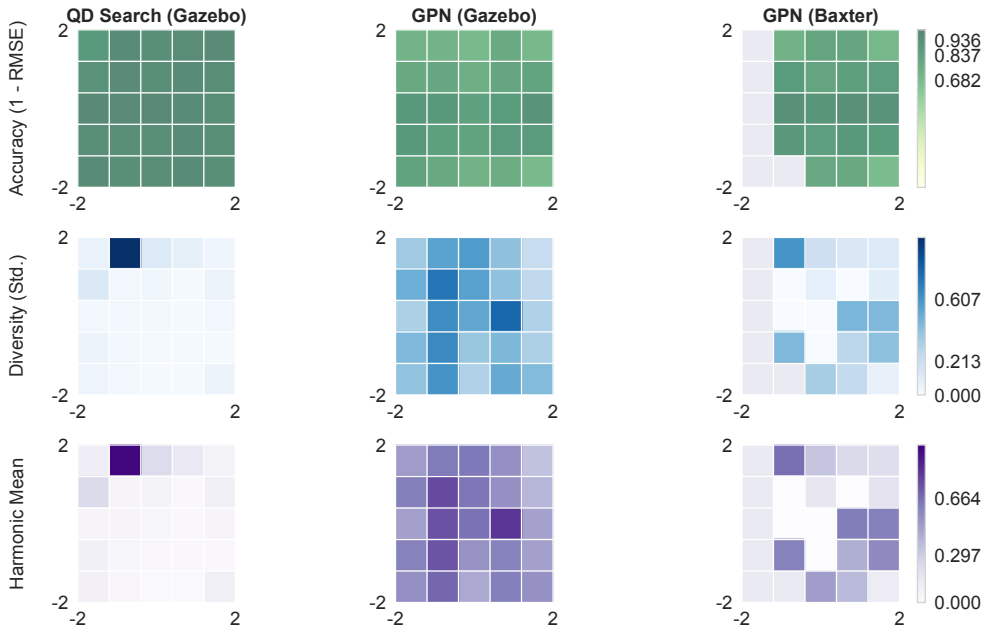

Fig. 4.4 Throwing to a 5×5 grid of points on the floor around the robot located at $(0,0)$ and facing right. Local throwing accuracy (top), diversity (middle), and their harmonic mean (bottom) when throwing by QD trajectories in simulation, GPN sampled controllers in simulation, GPN controllers on a real Baxter robot. GPN generally has higher diversity, and better overall performance (harmonic mean).

4.4.3 Experiment 2.1: Target-conditional throwing with obstacles: GPN vs QD

Setup Our motivating scenario was to use the learned conditional distribution over controllers to achieve robust throwing in the presence of obstacles. In this experiment, we evaluate this quantitatively using Gazebo Baxter simulator. Specifically, we consider a 5×5 grid of floor targets as before, and we throw to each of these targets with 10 diverse sampled controllers as before. For each of those throws, we simulate obstacles and calculate whether an attempted throwing trajectory fails due to robot or ball collision with the obstacle. To systematically explore these issues we run the simulation for $k = 1 \ldots 9$, $\tau = 0, 0.1, \ldots, 1.0$, and repeat for different occlusion rates $= 1\%, \ldots, 8\%$ when calculating $SuccessesProportion(k, \tau)$. For simplicity we model occlusions as a randomly selected set of inaccessible floor areas, where the total proportion of blocked floor area is the specified occlusion rate. We compute results averaging over a 5×5 target grid, 10 throws per target, and 1000 maps with random obstacle scenarios.

[2]Symbolic representations of significance levels in this paper are as follows: '+' stands for the method or parameter against which the rest are compared (usually GPN for methods or the parameter we are using for the parameter sensitivity assessment), '*' stands for significance level $\alpha = 0.05$, and '**' - for $\alpha = 0.01$.

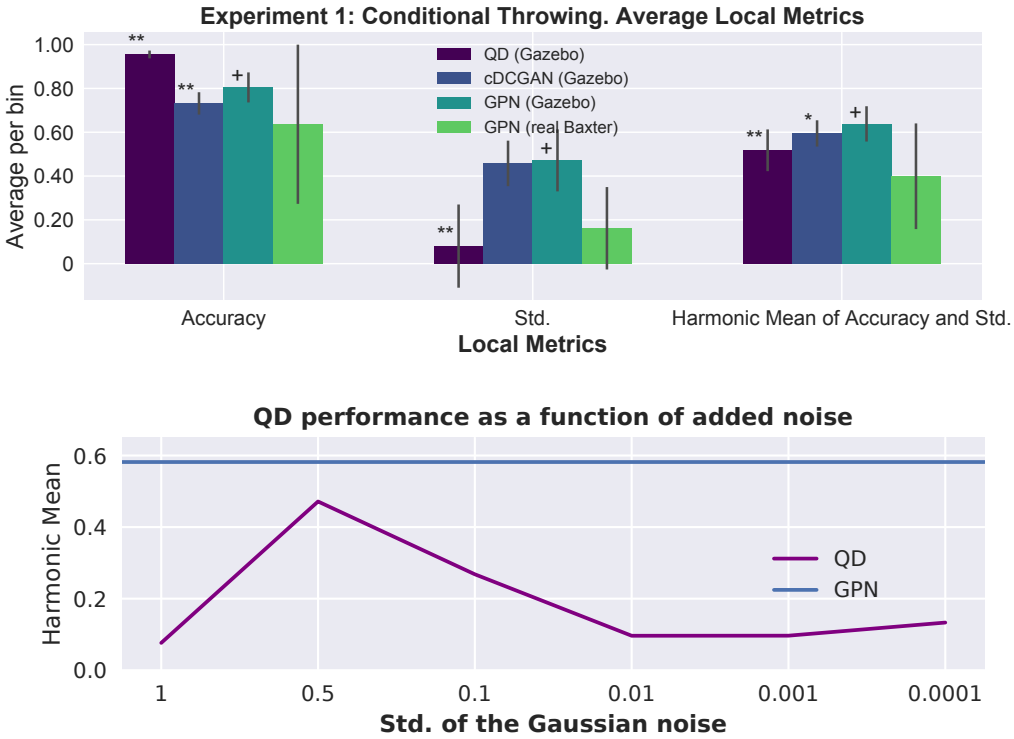

Fig. 4.5 Top: Summary statistics of throwing to all points on a 5×5 floor grid. The difference between harmonic means of the simulated data for QD and GPN is statistically significant according to unequal variances t-test with significance level $\alpha = 0.01$ $(p_{value} < 2.78 \cdot 10^{-12})$.[1] Bottom: Comparison to QD with varying levels of added noise.

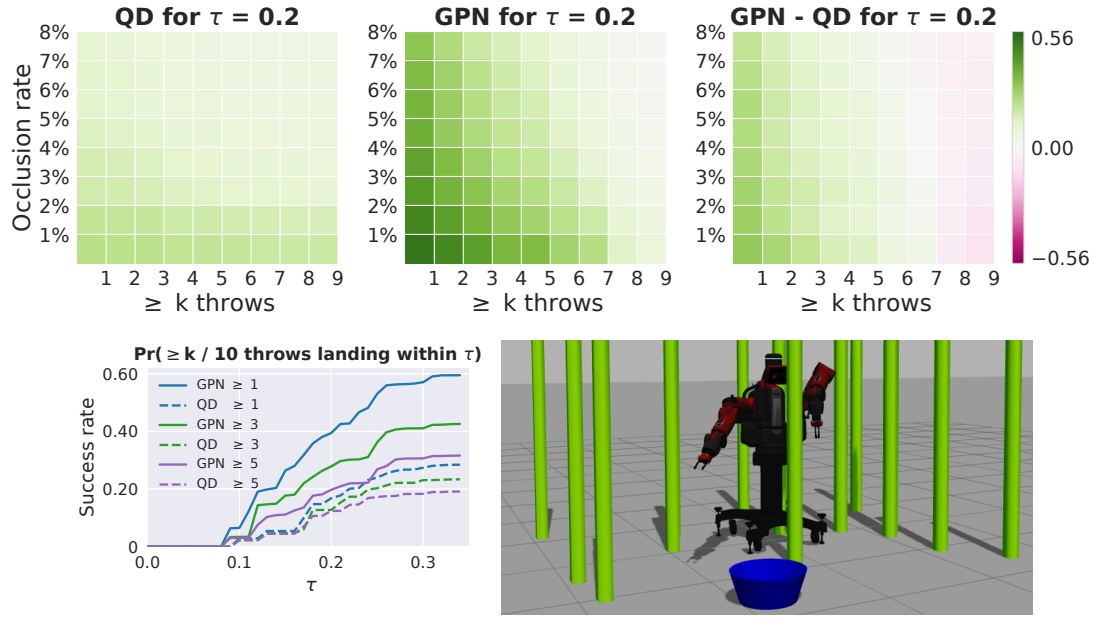

Fig. 4.6 Robustness of target-conditional throwing in obstacle-occluded environments. Top: Heat maps illustrate the probability of at least k of 10 throws landing within $\tau = 0.2$ of the target for different levels of occlusion. QD lookup method. Our GPN method. Difference between GPN and QD results. Bottom-left: success rate for varying accuracy requirements τ. Bottom-right: example of the random obstacle environment.

Results Figure 4.6 (bottom-right) shows the Gazebo simulation of Baxter attempting to throw to a specific target in the presence of randomly generated obstacles. Figure 4.6 (top) shows heat-maps of $SuccessesProportion(k, \tau = 0.2)$ for various occlusion rates and minimum hit requirements k. From these we can make the following observations: (i) Both QD and GPN methods have higher success rate in the easier bottom left (low occlusion, low hit ratio required for success), and vice-versa in the harder top right. (ii) The GPN result is much higher than that of QD for low k values (e.g., $k = 1$). This means that the GPN can often find at least one way to hit the target, for this whole range of occlusion rates. (iii) At very high minimum hit (e.g., $k = 9$) QD performance is slightly better than GPN. This is because GPNs slightly lower accuracy means that it's rarely the case that as many as 9 out of 10 attempts hit the target. However, at this stringent hit rate requirement, we note that the success rate of QD in absolute terms is also very low (around 10%). Finally, Figure 4.6 (bottom-left) shows the success rate averaged over occlusion rates as a function of different hit-radius requirements. We see that for a stringent accuracy requirement ($\tau < 0.1$), neither method succeeds. While for all larger values of τ, GPN consistently outperforms QD in success rate. Overall the results validate our goal: GPN can throw accurately enough that it often hits the target, but it does so in diverse enough ways that at least one way can usually be found to dodge any given obstacle configuration.

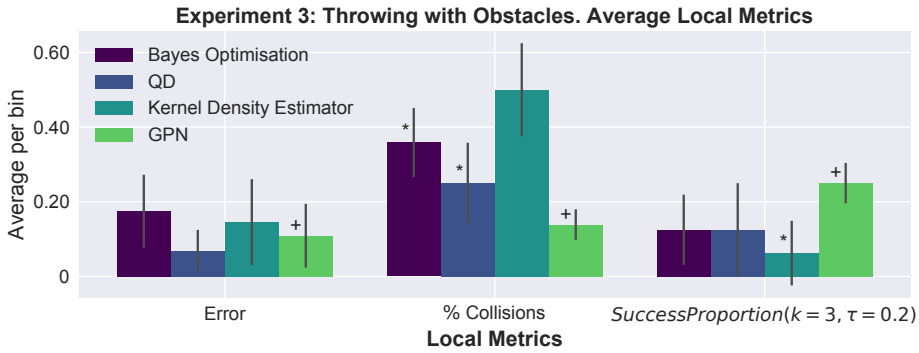

Fig. 4.7 Target-conditional throwing robust to obstacles. Comparison of GPN, QD, Bayesian Optimisation, and Kernel Density Estimation approaches. GPN is not the most accurate model, however is very low on collisions due to the diversity of its policies. Its greater diversity translates into higher *SuccessProportion* due to better dodging of random obstacles compared to other methods.[2]

4.4.4 Experiment 2.2: Target-conditional throwing with obstacles: Baseline Comparisons

Setup In this experiment, we compare two further alternative approaches to obstacle-robust throwing: KDE [47] and BayesOpt [163] (Section 4.3.3). We use a simulated setup where there is a wall randomly placed between Baxter and the target (see supplementary video and Section 4.4.5).

Results The results in Figure 4.7 average over four random goal/wall positions and compare the different methods in terms of error, diversity, collision rate and $SuccessesProportion(k = 3, \tau = 0.2)$. We see that while GPN is not the most accurate model, its success rate is best overall. This is because (i) it has good diversity enabling it to dodge obstacles more often, (ii) it has learned the manifold of reasonable policies, so it usually also avoids self-colliding or unsafe movements. In contrast, KDE and BayesOpt often generate self-colliding movements or hit the obstacle. Kernel Density Estimation suffers from being an inefficient/inaccurate model of relatively high-dimensional (15D) policies. Bayesian Optimization purposefully adapts the actuator movement to avoid the obstacle, but cannot succeed in the relatively small number (10) of available trials.

4.4.5 Experiment 2.3: Physical Baxter robust conditional throwing with obstacles

To test our method, we sample the conditional GPN ten times to generate ten diverse controllers that should throw to the required point. We simulate them to check for collisions with the obstacle, and found three of these avoided collision and landed into

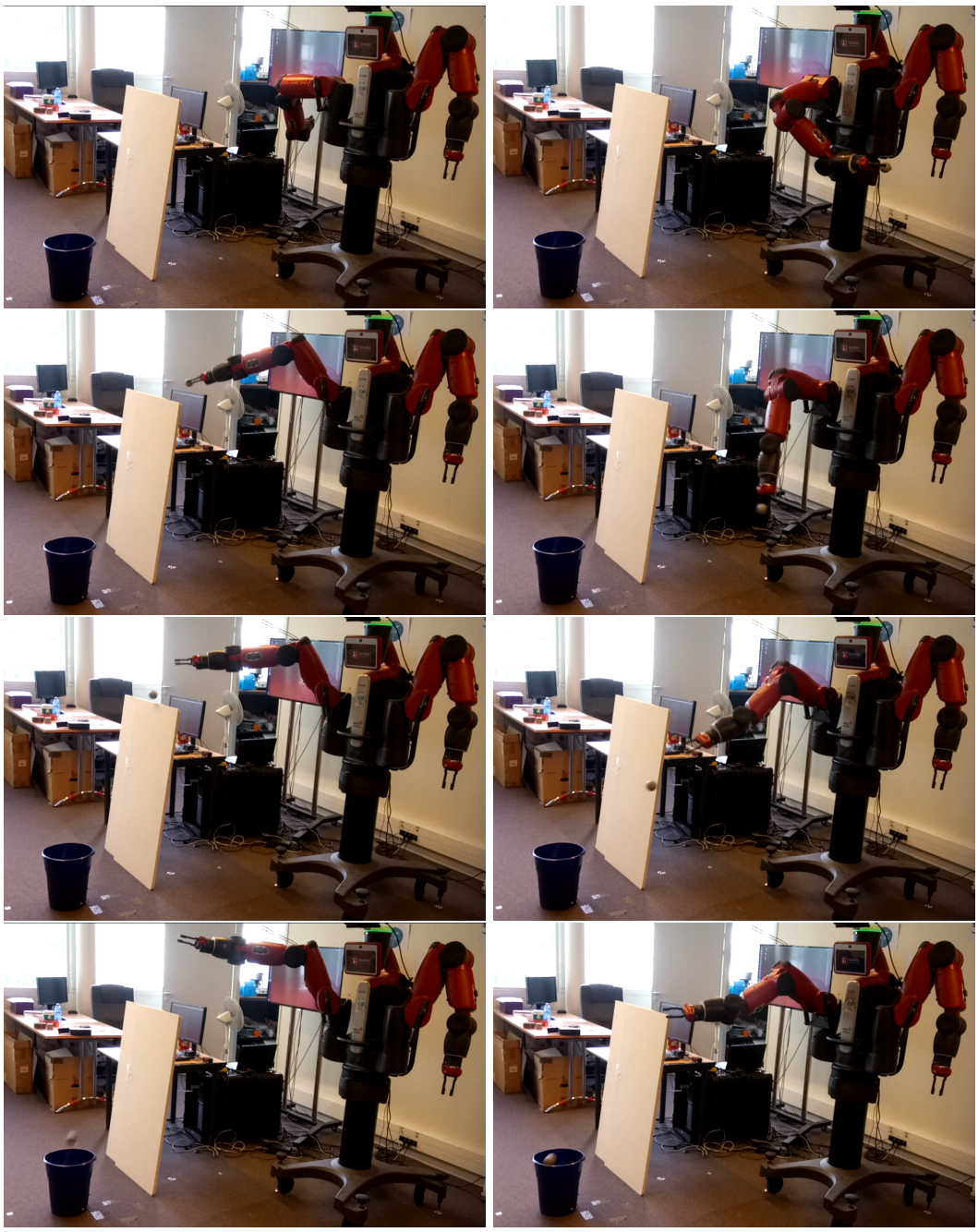

Fig. 4.8 Two examples of obstacle robust throwing behaviors obtained by sampling our learned distribution over policies. The GPN is conditioned on the target location, and samples controllers for throwing there until samples are drawn that generate neither robot nor ball collisions. For a video of this experiment please refer to the supplementary material: https://youtu.be/2LCnaa89erM

the basket in simulation. This validates that the GPN has indeed learned to generalize and samples novel behaviors. Figure 4.8 shows the successful execution of two of these controllers. We can see that the ability to generate diverse trajectories enables the robot to successfully hit the required target with the ball while dodging the given obstacle.[3]

4.4.6 Experiment 3: Throwing with damaged joints

Setup Another potential motivation for modeling behavioural diversity is to enable a robot to adapt its behaviour in response to physical damage [132]. In this experiment we explore this idea by evaluating the ability of GPN to find a behaviour that enables Baxter robot to complete its throwing task despite a hardware failure in one or more joints. There are various ways in which a joint can be damaged, for example: being jammed in a fixed position (unlikely in practice), full or partial joint sensor or actuator failure causing complete or semi-random motion. We are focusing on the latter. To simulate hardware sensor failure for a selected joint, random uniform noise is inserted to replace the planned velocity while performing a movement. This event would cause many (but perhaps not all) potential throwing plans to fail. So the aim is to show that GPN can generate diverse enough behaviours that at least some attempts succeed despite the hardware malfunction.

As before, the quantitative assessment has been run using Gazebo Baxter simulator. In this experiment Baxter is operating on a fixed subset of 5 targets from a 5×5 grid of floor targets as described in subsection 4.4.4. The controllers are sampled by the GPN from the target-conditional distribution learned from the training data. Baxter throws to each of the floor targets with 10 different controllers sampled by the GPN. For comparison we conduct the same set of experiments using the behavioural library assembled with the QD-search. The assessment metrics are near-identical to the subsection 4.4.3.

We assess the $SuccessesProportion(k, \tau = 0.2)$ in terms of how many throws can be land within τ distance of the desired target. The value of τ for the first set of results is selected arbitrarily and there are the further results are provided, demonstrating how the $SuccessesProportion(k, \tau)$ changes with respect to τ. In this experiment the $SuccessesProportion$ is expressed in terms of the of the number of successful throws for each individual damaged joint scenario. The Baxter robot has 7 joints, (denoted 1 to 7, where 1 is the closest to the body and 7 the closest to the end-effector, Figure 4.9, left). We repeat this experiment, simulating each joint being broken in turn, as well as some combinations of broken joints.

[3]For a video of this experiment please refer to the supplementary material: https://youtu.be/2LCnaa89erM

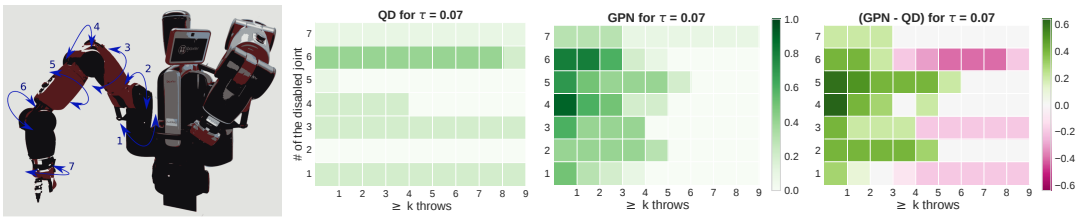

Fig. 4.9 Robustness of target-conditional throwing broken down by damaged joint. Left: the numbering of Baxter actuator joints. Here and in all the previous experiments Baxter only uses its right arm for throwing. Since it is symmetric, results should be the same for either side. Right: Heat maps illustrate the probability of at least k out of 10 throws landing within $\tau = 0.07$ of the target for different individual joints impeded. The first heat-map corresponds to the QD library lookup method, the second - to our GPN method, third shows the difference between the GPN and QD results. Our GPN sampling is equal or better than QD lookup for the majority of the scenarios, with the exception of larger k values for joints 1, 3, and 6. (Picture of Baxter (left) comes from [164].)

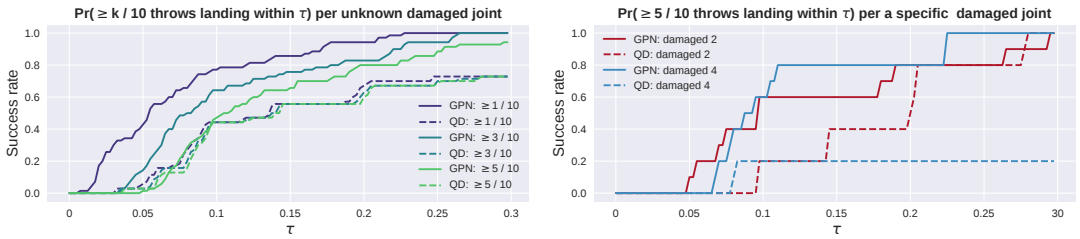

Fig. 4.10 Left: Throwing success rate with respect to the allowed distance from target τ, where 'success' counts as at least $k = 1, 3, 5$ minimum number of hits (out of 10) within radius τ. Average result across all single joint failures. Right: the success rates of half of the throw landing within τ of the target, for example damaged joints 2 and 4.

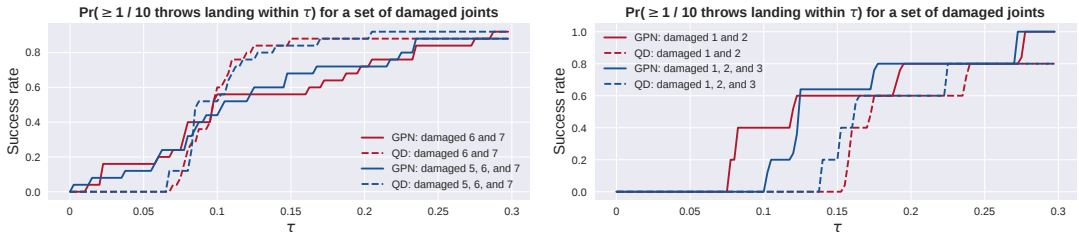

Fig. 4.11 Robust target-conditional throwing with multiple actuator joints disabled. Left: damaging 3 last actuator joints, one after the other, starting with the one closest to the end-effector. There is not much difference between the compared methods in this case: For very small values of τ, i.e. for a very precise throwing requirement our GPN is slightly better. However, for the less precise throwing requirement, QD is slightly better. Right: "damaging" three first actuator joints, starting with the closest one to the body. The data suggests that the GPN-sampled policies usually outperform the QD ones.

Results: Joint-wise Results are presented in Figures 4.9 and 4.10. First of all, similar to Figure 4.6 for the obstacle occlusion experiments we compare QD and GPN's ability to deal with impediments, in this case individual damaged joints. Evidently from Figure 4.9 heat-maps of $SuccessesProportion(k, \tau = 0.07)$, there is some variability in how well each model deals with individual broken joints, e.g., QD is relatively weaker for joint 2 and 7; GPN is weaker for joint 1. Nevertheless, in aggregate the GPN-sampled controllers are more often successful than QD ones, especially for the lower values of required minimum successful hits. The QD policies, however precise in the initial environment, provide no diversity of execution per fixed target. The diversity of the GPN policies translates into a higher chance of at least one out of ten of its policies overcoming the difficulty presented by an impaired joint.

Further, the following observations can be made: (i) the QD library is very evenly successful (or otherwise) over k for each damaged joint. This can be attributed to its high precision and low diversity - if the chosen trajectory happens to not be affected by the damaged joint very much and hits the target, it is likely to hit it for all 10 attempts. Unlike the averaging over the obstacle occlusion scenarios the stochasticity of the broken joints appears to represent a much more uniform constraint for the QD library. Thus, leading to performance less dependent on the minimal number of throws required. (ii) In contrast, the GPN offers a set of diverse solutions - so one of them being successful does not guarantee the success of the other ones - leading to relatively weaker performance at high k. (iii) Despite a small subset of scenarios where the QD library is more efficient, GPN shows comparable or better performance in the majority of cases, especially for the smaller amount of the minimal successful hits required (Figures 4.9, right). In principle, if one is not limited in the number of attempts, the GPN can be thus sampled until success is achieved.

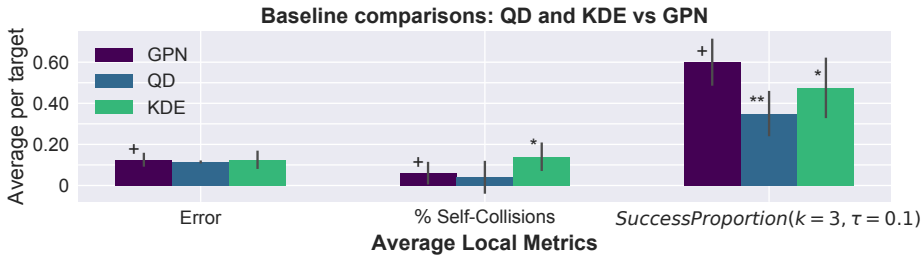

Fig. 4.12 **Baseline Comparison.**[3] Comparison of GPN, to QD and KDE, averaged across the multiple targets and damaged joints cases. All three have comparable Error rates, however the ratio of self-collisions is noticeably higher for KDE. The better success-rate of the GPN comes from the higher diversity of its controllers, overcoming the joints impediments.

The previous results are for an arbitrarily chosen accuracy threshold, τ. We next present a different view of the results by fixing the required hits k and varying the required accuracy τ. The plots in Figure 4.10 (right) show the probability of at least half of the executed throws landing in the basket, for a few example joints.

Results: Aggregate Finally, we present aggregate results averaged over all the potential damage points. Figure 4.10 (left) shows success rate as a function of accuracy threshold τ for some hit-rate requirements $k = 1,3,5$, when averaged across all the affected joint cases. Across all accuracy thresholds τ, the GPN policies are overall equally or more successful than QD at functioning successfully despite broken joints.

In summary, the results confirm that a GPN-based diversity strategy can successfully overcome joint failures for most of the scenarios considered. This in turn demonstrates the generalisation of GPN beyond the QD library upon which it was trained.

Baseline comparisons As in the experiments with the obstacle-occluded environment, we compare our GPN to alternative approaches in terms of aggregate statistics. Besides the library-based QD, we also consider sampling the Kernel Density Estimate defined over the same QD-based training set used by our GPN. The results presented next come from the same experiment above, averaging across the variety of damaged joint situations. As before we assess the methods in terms of the of error, diversity, collision rate and $SuccessesProportion(k = 3, \tau = 0.1)$. We conduct this experiment for one damaged joint at a time, performing 10 throws each to a selection of target points. Finally, we compute the average of each metric over all the target points and all the damaged joints.

The results in Figure 4.12 suggest that for throwing with an impeded joint, the average accuracy of the models is very similar. KDE exhibits a slightly higher self-

collision rate compared to the other two. And our GPN has the better Success Rate, facilitated by the higher diversity of its policies.

Multiple Damaged Joints A more challenging scenario is one in which multiple joints are simultaneously affected. We evaluate two settings. First is "damaging" the joints from the wrist joint 7 (in practice the most frequent to get damaged) up to the joint 5. Figure 4.11 (left) suggests that there is no clear difference between the compared methods. The second setting, is "damaging" joints starting with the joint 1 and going up to the joint 3. The results presented in Figure 4.11 (right) show that the GPN is usually equally or more successful than the QD in overcoming the multi-joint failures for the joints closer to the body.

4.4.7 Further Analysis

We finally evaluate the sensitivity of GPN to various hyper-parameters. Unless stated otherwise, the results in this section are averaged across test runs of 250 trajectories - 10 throws to 25 different targets, where targets remain the same for all the models to ensure the fair comparison[4].

Noise Parameters GAN-type models such as our GPN uniquely exploit a noise source to generate their diversity. We evaluate the impact of two parameters: the dimensionality and distribution of our latent noise source. Our experiments thus far were carried out with a $100D$ noise vector. Figure 4.13 (left) shows the results obtained from the same GPN configuration, but trained with noise inputs of various sizes. The results show that our 100D noise vector model exhibits the highest diversity, while maintaining one of the lowest error- and collision-rates. The larger 200D candidate also works well, implying that our model is not very sensitive to the size of the noise vector once it is above a minimum threshold. We also compared spherical versus uniformly distributed noise, but detected no clear difference in performance.

Training Epochs and Dataset Size Another factor to consider is the number of training epochs. The results in Figure 4.13 (right) show that performance generally improves with more epochs, suggesting that our GPN is converging to a good solution without overfitting or diverging. Notably, a sufficiently well-trained model produces nearly zero self-collisions.

Finally, we ask the question of how much data is necessary to train our model? The effect of training dataset size on the performance metrics is shown in Figure 4.14. The sizes compared are 1000, 5000, 10000, and 15000 (the total number of the training data

[4]Please note that unlike in the rest of the paper these results are based on training the GPN for 100 training epochs (for speed sake). This is justifiable as there is no statistically significant difference between 100 and 1000 training epochs according to Fig. 4.13, right.

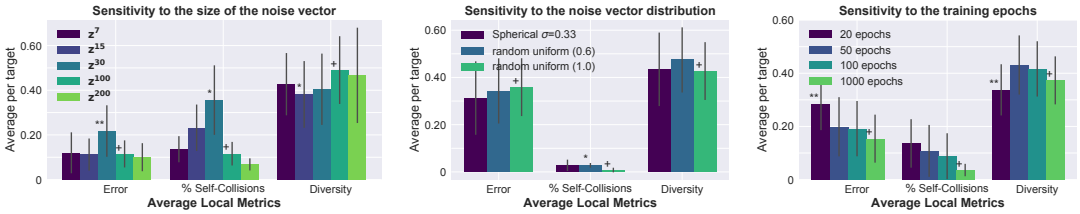

Fig. 4.13 **Parameter sensitivity.**[3] Left-to-right: **1.** Sensitivity of the GPN to the size of the generator noise vector in terms of the average local metrics (average error, ratio of self-collisions, and diversity of the trajectories).The GPN with 100D noise vector has been used for the rest of the results in this work, due to high diversity and low error. **2.** Sensitivity to distribution of the noise GPN noise vector. Both spherical and uniform random noise perform similarly, so we stick with the random uniform noise. **3.** GPN performance after various training epochs. The results confirm, that once trained sufficiently GPN produces next to no self-collisions.

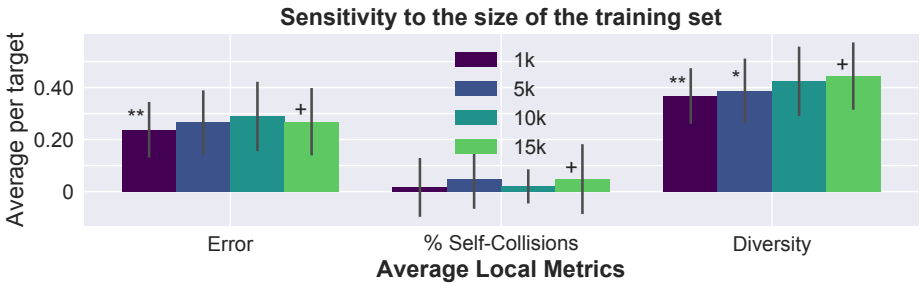

Fig. 4.14 **Dataset Size.**[3] Comparative performance of the same GPN configuration trained with datasets of different sizes. Despite the difference in the sizes of the training datasets, the performance is very comparable.

available). Performance is best with the largest 15000 sample dataset. However, we can see that it degrades relatively slowly with the decreasing amount of data, suggesting that the large scale training data is not crucial for our framework.

4.5 Discussion

4.5.1 Summary

To summarise, the results show that the GPN is capable of generating significantly more diverse conditional throwing policies compared to the initial training data at a small cost of accuracy (Experiment 1). This enables the GPN to successfully perform in two major applications - obstacle avoidance robust throwing (Experiment 2) and robust throwing in unknown broken joints setting (Experiment 3).

In case of the obstacle avoidance, we declare a success if at least $k/10$ trials results in a hit. Here the GPN approach is clearly better when a smaller amount of successful hits is acceptable. The lookup-table of QD data outperforms it when at least 80% success level required (Figure 4.6), because it will just repeat the same looked-up trajectory 10 times, hence $\approx 80-90\%$ success, *if* that trajectory happens to be successful in a particular obstacle setting. But in absolute terms both methods perform badly against this stringent requirement. These results are valid assuming no feedback is available about the result (hit or miss). If such feedback is available, the GPN could repeat its first successful trajectory in each setting, which will would further improve its success rates.

In the broken joints application, GPN similarly improves on QD when a smaller numbers of successful hits are required (Figure 4.9). It seems like QD policies are relatively more robust to certain joint failures (1, 3, and 6). However overall there is still an obvious benefit to using GPN in the settings of 1-3 unknown broken joints (Figures 8-12).

4.5.2 Significance

Overall our contribution fits in with and extends the successful existing line of work on learning repertoire-based robust behaviours [132–134, 162]. Rather than learning a fixed repertoire, where available diversity is set after training; we learn a conditional generative model for behaviours that can sample additional diversity online at run-time, and do so conditionally on a specified goal. This both allows increased robustness through greater available diversity, and potentially better scalability to very large repertoires due to carrying a parametric behaviour generator in place of an exhaustive behaviour library. Although we evaluated our technique on throwing, the methodology is generic and could be applied to any setting addressable by open-loop controller. Thus this provides a valuable tool that could help realise the vision of achieving robot robustness through behavioural diversity.

4.5.3 Future Work

In future, we intend to explore applying the proposed generative policy network framework for generating low-dimensional closed-loop controllers such as dynamic movement primitives [143] rather than our current open-loop controllers, and application to different kinds of tasks besides throwing [162]. Rather than relying on a fixed training set, we would also like to close the loop between generator learning and training set

collection, and gather data more efficiently by leveraging the GPN's ability to condition on a desired target.

4.6 Conclusion

We introduced the idea of generative policy networks, for defining a generative model over policies. We showed that our generative policy network provides a way to compactly encode a large set of known behaviors, and that sampling the GPN provides a way to draw unlimited novel controllers that are related-to but different-from known training behaviors. We showed how to apply this novel idea to provide an effective repertoire-based solution to challenging tasks including obstacle-robust throwing and joint-damage-robust throwing. This adds to the existing promising line of research on behavioural diversity and brings us one step closer to achieving robust behaviour through behavioural repertoire.

Chapter 5

SIDE-GANs for Trajectory Generation for Dynamic System Identification[1]

Dynamic System Identification approaches usually heavily rely on the evolutionary and gradient-based optimisation techniques to produce optimal excitation trajectories for determining the physical parameters of robot platforms. Current optimisation techniques tend to generate single trajectories. This is expensive, and intractable for longer trajectories, thus limiting their efficacy for system identification. We propose to tackle this issue by using multiple shorter cyclic trajectories, which can be generated in parallel, and subsequently combined together to achieve the same effect as a longer trajectory. Crucially, we show how to scale this approach even further by increasing the generation speed and quality of the dataset through the use of generative adversarial network (GAN) based architectures to produce a large databases of valid and diverse excitation trajectories. To the best of our knowledge, this is the first robotics work to explore system identification with multiple cyclic trajectories and to develop GAN-based techniques for scaleably producing excitation trajectories that are diverse in both control parameter and inertial parameter spaces. We show that our approach dramatically accelerates trajectory optimisation, while simultaneously providing more accurate system identification than the conventional approach.

[1]This chapter has been accepted for publication in IEEE International Conference on Intelligent Robots and Systems 2020.

5.1 Introduction

In the light of the continuous improvement in robotic mechanical design, the importance of accurate models for robot dynamics increases immensely. Model inaccuracies can have a significant effect on control, stability, and motion optimisation of the platforms. Hence the problem of system identification in robotics is currently being revisited. All dynamics system identification techniques are highly data-dependent in the sense that their parameter estimation efficacy, and the generalisation of dynamics models using these parameters, are highly affected by the quality of the trajectories used for exploration.

Most conventional system identification methods rely on a single parameterised trajectory [165–167]. Such trajectories are limited in how much they can explore system parameters within their set length. Extending the length of this trajectory alleviates this limitation, however the computation required to generate an optimal excitation trajectory grows rapidly with trajectory length, and quickly becomes intractable. We explore whether generating multiple shorter diverse trajectories can be used to achieve the same effect more scaleably than existing methods – or to outperform them – by effectively allowing the generation of longer overall trajectories. If many shorter trajectories, diverse in the inertial parameter space, can be generated, they can ultimately better explore all the parameters than a single longer trajectory, while being easy and cheap to generate in parallel. Furthermore, assuming that they are cyclic (same start and end condition), the set of short trajectories can be concatenated for easy execution in sequence. We provide experimental results to confirm that generating multiple shorter excitation trajectories tends to be better than the equivalently long single excitation trajectory in terms of system identification performance.

To fully leverage this paradigm of system identification trajectory generation, we need the ability to efficiently generate numerous diverse short excitation trajectories. To this end, we propose a pipeline where a traditional trajectory optimizer is used to generate an initial seed dataset, after which we train a Generative Adversarial Network (GAN) on this seed set[2]. Once trained the GAN provide a surrogate model for excitation trajectory optimisation that can effectively generate an unlimited number of diverse short excitation trajectories rapidly and in parallel. Our generative model is optimised with respect to the validity (in terms of the constraints and avoiding self-collisions) and fitness (excitation) scores, in order to generate a dataset that is maximally informative about system dynamics.

[2]Here and later "seed set" stands for the original training set of shorter informative trajectories.

Our proposed method is called System IDEntification GAN (SIDE-GAN). In principle, the SIDE-GAN is indifferent to the dynamics model behind the training trajectories. So, given the suitable initial training dataset, SIDE-GAN can provide quality training data for either parametric, semi-parametric, or even for some non-parametric system models. Our empirical results show that our SIDE-GAN approach improves the accuracy of parameter estimation and torque prediction, with a recursive least squares identified model. Furthermore, our generation speed increases by over two magnitudes compared to the original short trajectories optimisation, and allow us to generate a total excitation trajectory length that is intractable with traditional long-trajectory optimisation. Our empirical results are demonstrated using both real and simulated KUKA LWR IV manipulation platforms.

5.2 Related Work

Current models for dynamics system identification. Previous work on generating exciting trajectories for inertial parameter identification has mainly focused on approaches that optimize a single parameterized trajectory for optimal excitation [165–167]. Typically, this is achieved by maximizing the identifiability of each parameter via the stacked regressor matrix from the trajectory that is being optimised (please refer to the Section 5.3 for more details).

The optimization metrics tend to be highly non-linear with even more complex constraints, such as avoiding self-collision or certain regions of space completely. The optimization task is thus extremely difficult due to non-smoothness and many local minima. Genetic algorithms [167] are often used, but take many iterations to converge. Moreover, the cost of evaluating a single step of this optimizations is more than $O(n^3)$ in the desired length of the trajectory.

To our knowledge no previous work has explored optimizing multiple trajectories. Our divide-and-conquer approach can generate longer (and thus more informative) trajectories than the traditional approach. Besides being over two orders of magnitude faster, we find that our multi-trajectory can outperform the standard approach, even when controlling for the total length of the final trajectory.

Generative Adversarial Networks GANs [4] are a family of neural network methods, that have gained popularity for realistic image generation since 2014. They have since been applied to a multitude of tasks, although their primarily focus has largely remained realistic image and video generation [83, 55, 88, 59, 89], for example from captions.

Compared to the image generation area, there is still comparatively little research on how GANs can be of significant help in the field of robotics. Conventional image

GANs can generate data to assist training visual recognition systems in autonomous systems [6], however applications of GANs to planning and control are still very sparse.

The most relevant GAN-based approaches to non-sensory part of robotics so far include imitation learning [63, 64] – to efficiently learn a single policy or a discrete set of policies from demonstration; and direct generation of robot control policy repertoires [5]. The latter provides robust goal-directed control by enabling sampling diverse controllers from a continuous goal-conditional distribution over control policies. There has also been some research conducted on learning the inverse kinematics (IK) of the robot using GAN-like architectures [168]. However this work does not leverage the diversity potential of GANs, discarding the random noise input completely, instead replacing it with the end-effector position (for the IK problem). This replacement strips the generator of all the diverse generative properties, boiling it down to merely a mapping network, and the whole architecture to an actor-critic-like model.

We provide the first application of GAN-like methods to the 'experimental design' aspect of dynamics system identification. We learn our SIDE-GAN that on a seed set of excitation trajectories, which then provides a surrogate model to replace the typical compute intensive trajectory optimisation process. We rely on the ability of the trained GAN to rapidly generate *diverse* cyclic trajectories which can then be combined to provide an informative long trajectory.

5.3 Problem and Motivation

Dynamics system identification is the task of learning the inertial parameters π of the links of the robot. Normally this is achieved by starting with the Rigid Body Dynamics (RBD) equation:

$$\tau = \mathbf{M}(q)\ddot{q} + \mathbf{C}(q,\dot{q})\dot{q} + G(q) + F(q,\dot{q}) \tag{5.1}$$

Where $\mathbf{M}$ represents the inertia matrix, $\mathbf{C}$ the Coriolis and centrifugal matrix, G is the gravity vector, F is the friction vector and τ is the full joint torques experienced and measured by the robotic platform. The state of the robot is expressed in terms of $(q,\dot{q},\ddot{q})$ - position, velocity, and acceleration. We then use the RBD equation and rearrange it to form Eq. (5.2), which is linear with respect to π. This allows us to the use standard least squares approach [169] to solve for π, shown in (5.3).

$$\tau = Y(q,\dot{q},\ddot{q})\pi \tag{5.2}$$

$$Y^{-1}(q,\dot{q},\ddot{q})\tau = \pi \tag{5.3}$$

This solution would be valid for a single state of the robot but would rarely be the correct model due to noise in the data. Typically, the dynamics is sampled at many

different input states (positions, velocities, and accelerations) to compensate for the noise. This allows multiple regressors to be constructed in a stacked matrix, alongside their equivalent stacked torque vector:

$$Y = \begin{bmatrix} Y(q_0, \dot{q}_0, \ddot{q}_0) \\ ... \\ Y(q_n, \dot{q}_n, \ddot{q}_n) \end{bmatrix} \quad (5.4) \qquad \tau_s = \begin{bmatrix} \tau_0 \\ ... \\ \tau_n \end{bmatrix} \quad (5.5)$$

We can then estimate the inertial parameters π using the pseudo-inverse of the regressor matrix:

$$Y^T(YY^T)^{-1}\tau_s = \pi \quad (5.6)$$

Numerical errors can occur if YY^T is ill-conditioned (e.g., has very small or zero eigenvalues). When regressors are ill-conditioned, the trajectories sampled to provide state-torque pairs do not sufficiently excite the relevant parameters. For example, a sampled trajectory may not accelerate enough for the inertia to affect the output torques. With low excitation these parameters are not identifiable from the sampled data, and the regressor will be ill-conditioned.

Discussion: Non-identifiable parameters Please note that some parameters are always non-identifiable as they have no affect on the output torque no matter the state of the robot. These non-identifiable parameters can be removed by calculating the base parameter set of the robot [170] which will let us replace the regressor matrices with a base regressor matrix, Y_b, and replace the inertial parameter with the base inertial parameters, π_b, which contain the set of parameters that are excitable, such that (5.7) holds. From this point forward when the stacked regressor Y is referred to, it contains the base regressors of each state rather than the full regressor, and with π replaced by π_b.

$$\tau = Y_b(q, \dot{q}, \ddot{q})\pi_b \quad (5.7)$$

Quantifying Trajectory Excitation As discussed earlier, unexciting trajectories can lead to base regressors that are still low rank with low excitation, meaning ill-conditioned YY^T and numerical errors in Eq. (5.6). The goal of trajectory optimization is to produce trajectories $((q_0, \dot{q}_0, \ddot{q}_0, \tau_0), \ldots, (q_n, \dot{q}_n, \ddot{q}_n, \tau_n))$ that lead to well-conditioned YY^T and accurate estimation of parameters π. There are two different objectives correlated with the trajectory quality, that can be used during the SIDE-GAN training. In section 5.5.4 we show that either can be used for training the SIDE-GAN, and results are comparable:
Eigenvalue Fitness: The first metric is condition number, i.e., minimal ratio between the largest and smallest eigenvalues of Y^TY. This implies a high parameter excitement *within the trajectory*, with the least and the most excited inertial parameters being

explored as equally as possible, directly leading to a better conditioned YY^T matrix. We refer to the condition number of Y^TY as 'fitness', and use it to report the progress of training in Figure 5.3.

Diagonal Fitness: Another objective is based on the trace of Y^TY, i.e. either maximising the trace itself or minimizing the trace of the inverse. The diagonal of Y^TY is indicative of the level of exploration trajectory does in the inertial parameter space. The trace is indifferent to basis changes and hence is more comparable *across the trajectories* with different basis vectors. We refer to

$$\frac{\max(\text{diag}(Y^TY))}{\min(\text{diag}(Y^TY))} \tag{5.8}$$

as the 'diagonal fitness', and use it for training one of the two versions of the SIDE-GAN assessed in Section 5.5.4. Similarly to the conventional fitness, smaller scores correspond to more evenly explored inertial parameters.

System Identification The conventional approach to dynamics system identification optimizes for a trajectory with high fitness, executes this trajectory on the robot, and then estimates inertial parameters as in Eq. 5.6. However, given the cost of optimizing long trajectories, the achievable fitness and parameter identification accuracy is limited.

Use of Multiple Trajectories The unique aspect of our approach is to use multiple trajectories to improve inertial parameter exploration. For the purpose of system identification, we desire not only diversity in control parameter space, but also to explore different subsets of inertial parameters. Then their stacked regressors combined will have more uniformly distributed eigenvalues, and when inverted will produce a better estimate of inertial parameters π. The eigenvalues of the stack of short trajectories correspond to inertial parameter identifiability, as for the conventional single-trajectory approach. We use Modified Fourier Trajectories [166], which are cyclic in the start and end conditions, making concatenation of several trajectories together trivial. We investigate: Whether a sufficiently large and diverse set of cyclic trajectories can be generated? And how this set of diverse trajectories performs for system identification compared to a standard single longer trajectory?

Using multiple short trajectories also has an important advantage over a single long one in terms of generation efficiency. The time of optimizing the trajectories grows quickly as the trajectories grow longer, due to the $O(n^3)$ complexity computing Y^TY and the additional complexity of computing the self-collisions at each discrete step. Optimising multiple short trajectories means much faster regressor calculation, as well as enabling trivial parallelisation.

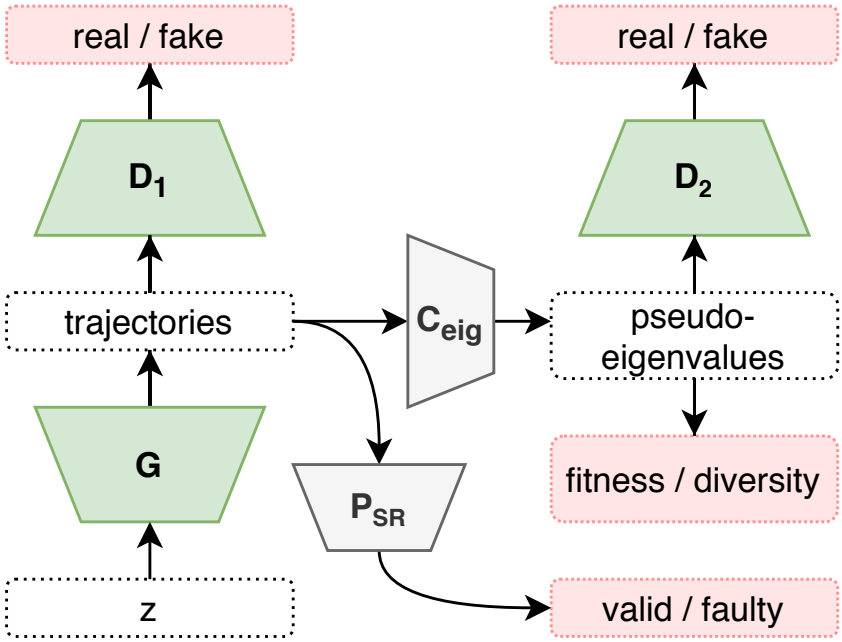

Fig. 5.1 **SIDE-GANs at training time:** In the conventional GAN architecture on the left, the generator G inputs a noise vector z, and outputs synthetic trajectories. The discriminator D_1 tries to distinguish "fake" (synthetic) vs real trajectories.
The right side of the scheme represents the new part of the system, where the pre-trained predictor P_{SR} predicts valid/faulty, and thus provides a penalty for invalid trajectories. The pre-trained converter C_{eig} maps trajectories to their estimated eigenvalues. These pseudo-eigenvalues are then assessed by the second discriminator D_2 as "real" or "fake", they are also used as input to compute an eigenvalue fitness penalty that encourages high excitation trajectories.
Colour-coding: Trapezoid blocks are neural networks. Grey trapezoids correspond to pre-trained networks with weights frozen for the main SIDE-GAN system training. Green trapezoids correspond to the networks that are learned during main system training. Pink blocks are used during training to compute losses.
Please refer to Figure 5.2 for more details on architerture of each of the networks.

Summary Our proposed pipeline uses conventional optimisation to generate a set of short trajectories, then trains SIDE-GAN to to rapidly expand this dataset. The ultimate testing objective is then to show that model-based torque prediction (using parameters obtained from system identification) is more accurate when using the SIDE-GAN-generated trajectories than based on just the initial seed data, or the conventional single longer trajectory.

5.4 Method and Architecture

Architecture SIDE-GAN is built for the generation of diverse excitation trajectories for system identification. Figure 5.1 and 5.2 show the architecture and describe the

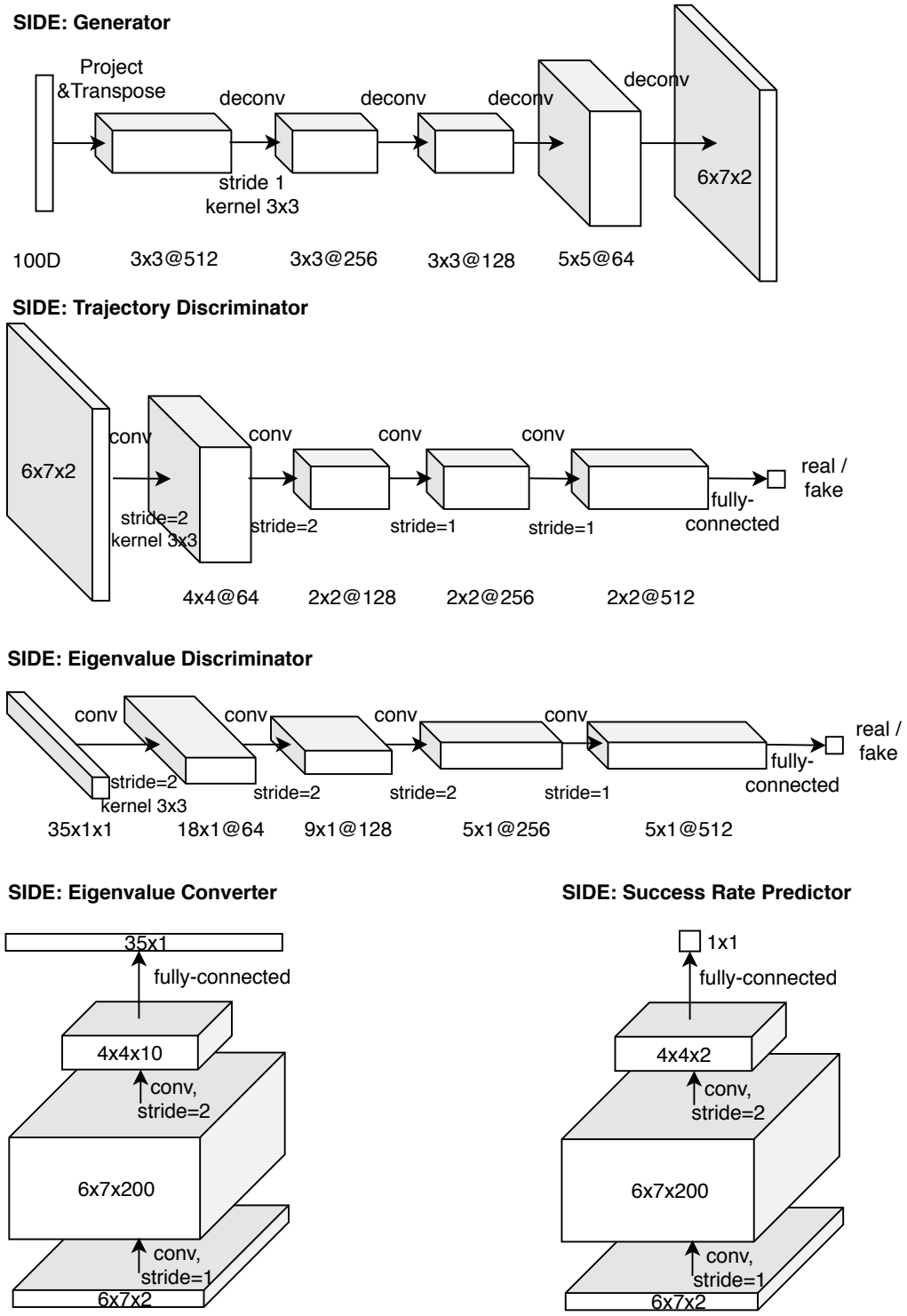

Fig. 5.2 **SIDE-GANs Networks architecture (elaboration on Figure 5.1) top-to-bottom:** generator G, trajectory discriminator D_1 - checks the realism of trajectory parameters, eigenvalue discriminator D_2 - assesses the realism of pseudo eigen-values, Converter C_{eig} - converts trajectory parameters into pseudo eigen-values (pre-trained for 350 epochs), and success rate predictor P_{SR} (or more of a trajectory validity predictor, pre-trained for 500 epochs). Then the rest of the system (G, D_1, D_2) is trained for 50 epochs with batch size 100.
Padding on all conv and deconv layers is 1, all the conv layers have leaky ReLu activation functions and deconv layers have ReLu activations.
Adam is used as an optimiser for all the networks with usual parameters - learning rate 0.0002 and $\beta_1 = 0.5$.

training details. SIDE-GAN is trained on a set of seed trajectories generated by the conventional optimizer, and once trained can rapidly generate new trajectory batches.

We build our model on the top of a typical Deep Convolutional Generative Adversarial Network architecture, i.e., DCGANs [2]. While conventional DCGANs generate images, we modify them to generate $7 \times 6 \times 2$ tensors representing fourier transform parameters, which define short cyclic trajectories. For our goal of system identification, there are a number of extensions required to adapt DCGAN to generate trajectories that are valid (e.g. non-colliding) and diverse in both control parameter and regressor eigenvalue (inertial parameter) space. These are detailed as follows:

Success Predictor Loss: Vanilla GAN does not ensure that the majority of generations are valid (i.e., non-self colliding, or constraint-violating). To address this, we define a success predictor, as a shallow convolutional network mapping generated trajectories to a valid/faulty label. We pre-train this network to differentiate the initial dataset of valid trajectories, and some invalid trajectories generated by vanilla GAN. The trained success predictor has 99% validation accuracy. We fix its weights and use it as a loss for the main SIDE-GAN training, thus encouraging it to generate valid trajectories.

Trajectory-to-eigenvalues converter The salient feature space for analysis of trajectories is the eigenvalues of the rolled-out trajectory. To predict these features for generated trajectories, we pre-train a shallow convolutional network to map fourier trajectory parameters to the resulting eigenvalues. As above, we freeze its weights before plugging it into SIDE-GAN. The estimated eigenvalues are then used by the following two modules:

Eigenvalue Discriminator. The basic GAN discriminator differentiates real vs fake trajectories in the GAN's raw output space (modified Fourier parameter tensors). However, for our purposes, the crucial property of the trajectories is to cover the eigenvalue space well. Therefore we define a second discriminator that differentiates real/fake samples based on the eigenvalues of the rolled out trajectories – as predicted by the eigenvalue estimator defined above. This is learned jointly with the vanilla discriminator in SIDE-GAN.

Fitness Loss. SIDE-GAN so far aims to generate trajectories that are valid, and indistinguishable from the seed set used for training. Nevertheless, other things being equal, for SIDE purposes, we prefer trajectories with a more uniform eigenvalue distributions. We therefore define a final loss that penalizes the eigenvalue fitness (Sec III). This is trained jointly with the other SIDE-GAN modules, but activated after epoch 10 once the rest of the model has stabilized.

Alternative Diagonal Architecture. The above three modules (converter, second discriminator, fitness loss) are based on eigenvalues. We also compare an alternative

approach based on diagonal Fitness (Sec III). In this case the converter estimates the Y^TY diagonal, the discriminator discriminates based on this diagonal, and the fitness is defined as in Eq. 5.8.

Training SIDE-GAN the generator produces a batch of trajectories defined by 'fake' modified Fourier transform parameters. These are mixed with the real trajectories, and the first discriminator labels these are real or fake, and the success predictor labels them as valid or faulty. The converter translates trajectories into pseudo-eigenvalues, which are then used by the second discriminator for labelling as 'real' or 'fake'. Finally, the pseudo-eigenvalues are also used to calculate and penalize the eigenvalue fitness metric. The training objective of SIDE-GAN is to produce diverse trajectories with small fitness loss and good validity scores.

The full training objective has the following form:

$$\begin{aligned}\min_{G_\theta}\max_{D_{1\phi},D_{2\psi}} V(G_\theta,D_{1\phi},D_{2\psi}) &= \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D_{1\phi}(\mathbf{x})+\log D_{2\psi}(C_{eig}(\mathbf{x}))]\\ &+\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1-D_{1\phi}(G_\theta(\mathbf{z})))+\log(1-D_{2\psi}(C_{eig}(G_\theta(\mathbf{z}))))]\\ &+\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\lambda P_{SR}(G_\theta(\mathbf{z}))+\gamma F(C_{eig}(G_\theta(\mathbf{z})))]\end{aligned} \tag{5.9}$$

where $\mathbf{x}$ stands for a data example, $\mathbf{z}$ - a random noise vector, $G_\theta(\mathbf{z})$ is a sample from the generator, $D_{1\phi}(\mathbf{x})$ and $D_{2\psi}(C_{eig}(\mathbf{x}))$ represent the discriminators estimate of the probability that $\mathbf{x}$ came from the real data set rather than from the generator, and $D_{1\phi}(G_\theta(\mathbf{z}))$ and $D_{2\psi}(C_{eig}(G_\theta(\mathbf{z})))$ - a probability that the data came from the generator. θ,ϕ,ψ - are the network parameters for generator and both discriminators correspondingly. $P_{SR}(G_\theta(\mathbf{z}))$ represents the predicted success rates of the generator output (i.e., the proportion of the valid trajectories amongst the generated data). $F(C_{eig}(G_\theta(\mathbf{z})))$ is the predicted fitness of the generator output. The coefficients (deduced empirically) λ and γ were set to 1 and 0 correspondingly for the first 10 epochs and then $\lambda=(1+0.06*i)$, where i is the number of the training epoch and $\gamma=2$. Setting γ to zero originally offers SIDE-GANs an opportunity to learn producing valid (non-colliding and non-self-colliding) trajectories first, and then shifting the focus towards improving the fitness of the generated trajectories.

We use the seed set to train the SIDE-GAN for 50 epochs in total (generator and both discriminators, as eigenvalue converter and success rate predictor are pre-trained, as specified in Figure 5.2), using two generator iterations for each iteration of discriminator cycle, and then discard everything but the generator.

Applying SIDE-GAN: The trained generator network provides our surrogate model for trajectory generation. It can generate as many diverse, new and exciting trajectories as

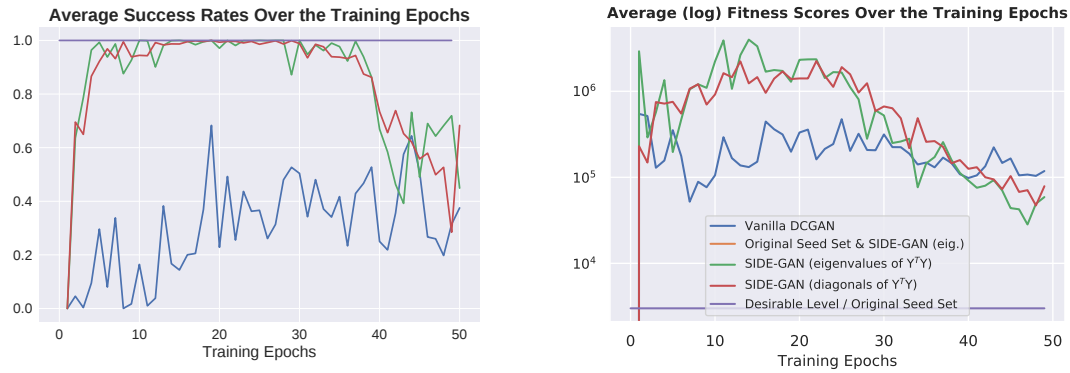

Fig. 5.3 **Average metrics over the SIDE-GAN training time:** the SIDE-GAN is trained for 50 epochs in total, the averages are taken over 3000 trajectories per epoch for each of the methods, i.e., 3 training runs, 10 noise vectors, 100 trajectories (batch size) produced by each input noise vector. **1.** Average success rate - i.e. ratio of the valid trajectories SIDE-GAN generates on average. **2.** Average fitness - i.e., the average condition number of the Y^TY for the entire trajectory. The lower the fitness score, the better is the quality of the trajectory for the system identification purposes. The fitness scores are very high in the beginning of training, so logarithmic scale is used for fitness score here.

necessary. Since it generates optimization-free, in a single forward pass, it can produce novel trajectories near instantaneously compared to traditional optimisation.

5.5 Experiments

5.5.1 Training Data & Metrics

The data we use for training SIDE-GAN were acquired using a genetic optimizer for a rough global solution (ant colony based [171]) followed by a finite difference gradient-based solver to fine-tune the trajectory locally [172], using the pagmo2 library [173]. The dynamics and regressors for the task were computed through the ARDL library [174]. The full seed training set consists of 1800 cyclic trajectories of 16 seconds each, that are represented in terms of $7 \times 6 \times 2$ tensors or parameters of the modified Fourier transform (these correspond to the number of manipulator joints, the number of Fourier transform parameters, and 2 points defining the cyclic trajectory). Corresponding 35 eigenvalues of the resulting trajectory are used during the training to optimise the quality of the trajectories in terms of inertial parameter exploration.

All of the performance results in this paper are averaged across at least three complete trainings of SIDE-GANs from scratch, and 10 batches of trajectories produced from different noise vectors.

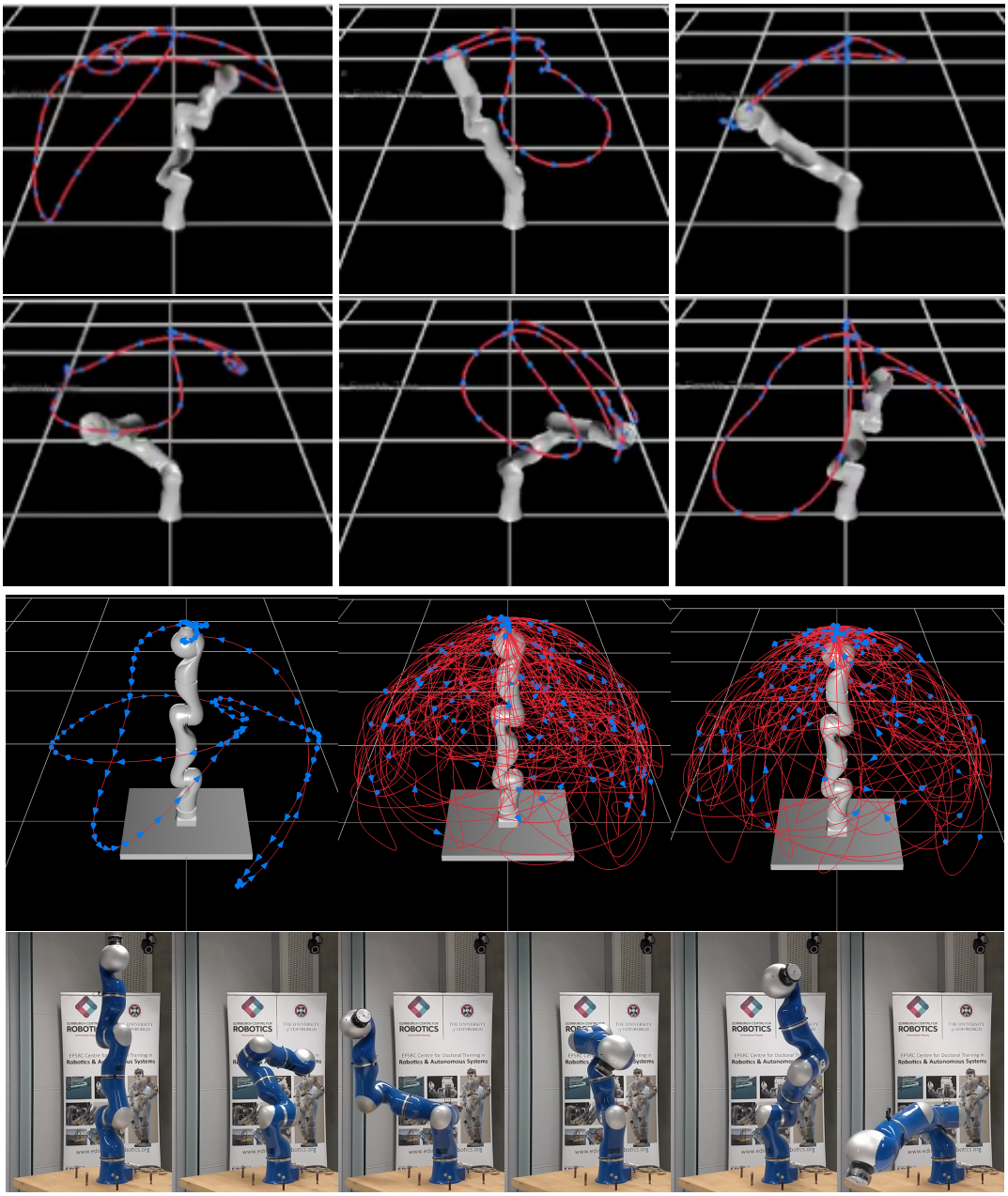

Fig. 5.4 **Visual results: 2 top rows:** examples of the SIDE-GAN generated trajectories - red traces show the end-effector positions. Trajectories are clearly spatially diverse. **Middle row:** examples of the single 560s trajectory vs. the original seed set and the SIDE-GANs $35 \times 16s$ trajectories stacked together. **Bottom row:** a few snapshots of one of the 16s SIDE-GAN trajectories.

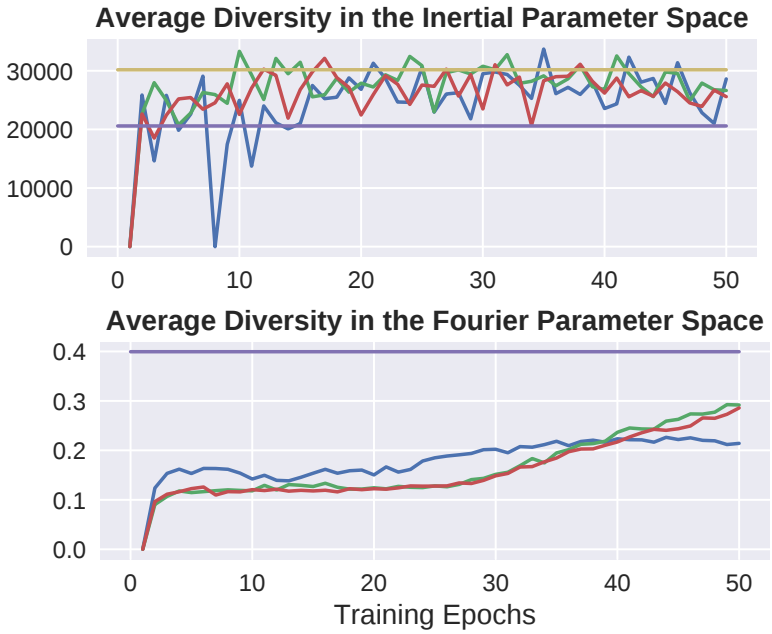

Fig. 5.5 **Average metrics over the SIDE-GAN training time:** The diversity of the generated batch is assessed via the average pairwise Euclidean distance between trajectories in two spaces of interest. The bottom: diversity in the Fourier parameter space (immediate output of the generator) - at epochs 40-50 the SIDE-GAN overtakes vanilla DCGANs. The top: the average pairwise Euclidean distance between $\text{diag}(Y^T Y)$ of the trajectories in the generated batches, representing the batch-diversity in the inertial parameters space. The SIDE-GANs diversity is usually equal or higher than that of the vanilla DCGANs, significantly surpassing the original seed set diversity (purple). The average inertial parameter diversity of the SIDE-GAN data (generated after 50 training epochs) and original training set put together (yellow) shows that adding the data synthesized by the SIDE-GAN to the original dataset is highly beneficial.

5.5.2 SIDE-GAN Training

We first answer the question: '*Can a neural network learn to generate valid, exciting, novel, and diverse trajectories?* To answer this, we analyse SIDE-GAN training dynamics in terms of success rate[3], fitness (as defined in Sec 5.3), and the diversity of trajectories within a generated batch. Generation of trajectories that exhibit diversity between themselves is necessary, because if a trajectory generator simply repeats itself, little new information will be added when short trajectories are combined into a longer one for execution. For diversity, we use two metrics: (1) Average euclidean distance between raw trajectory Fourier parameters, (2) Average euclidean distance between the $\text{diag}(Y^T Y)$ vector of each trajectory. These metrics thus cover diversity in both spatial and inertial parameter perspectives.

[3]Ratio of valid generated trajectories. I.e. non-self-colliding, obeying the joint velocity and position constraints, as well as the physical space constraints (e.g. avoiding the area within 10 cm above the desk on which the manipulator is set up).

Figures 5.3 and 5.5 shows the training dynamics of SIDE-GAN using the 1800 seed trajectories as training data, and compares vanilla DCGAN with our two (eigenvalue, and diagonal-fitness) variants.

From the plots we can see that training dynamics are somewhat unstable as per-usual with GANs: (1) Both SIDE-GAN variants quickly learn to reliably generate successful trajectories, while vanilla DCGAN struggles to pass 40% success rate. (2) Vanilla DCGAN initially generates better fitness than SIDE-GAN, but by later epochs (40-50) SIDE-GAN produces better fitness. (3) In terms of the batch diversity, after 40 training epochs SIDE-GANs have greater spatial diversity than vanilla DCGAN and comparable overall spacial diversity to the original seed set. In inertial parameter space SIDE-GANs are significantly more diverse than the original seed set and usually equal or better than vanilla DCGAN. We generate 4200 novel trajectories from SIDE-GAN after 50 epochs, and together with the initial data, this provides a total set of 6000 short trajectories which are used in the following system identification experiments. Figure 5.4 visualizes a few sample trajectories from SIDE-GAN. This visualisation shows that they are very diverse, at least in trajectory parameter space.

We also tried training some conventional non-neural network generative models such as Kernel Density Estimatiors (KDE)s. However, lacking a discriminator to provide a strong objective, these completely failed to produce valid trajectories, with an overall success rate under 0.1%.

5.5.3 Exp 1: Multi- vs Single-Trajectory System Identification

We first evaluate our idea of generating a set of smaller trajectories against standard practice of optimising the longest single trajectory that is computationally feasible. We stress that the key fundamental advantage our approach is: (1) Scalability to effectively unlimited total length, and thus much greater total excitation, unlike single trajectories. (2) Enabling massive parallelization for fast generation at any scale. Nevertheless, for the purpose of this experiment, we put these points aside and focus on comparing identification performance using a *fixed* total length – generated by a single trajectory optimisation, or our multi-trajectory optimisation.

Setup: Dataset To compare the trajectory optimisation methods, controlling for trajectory length, we optimise a single long trajectory of 560 seconds (the longest we can feasibly optimise) using eigenvalue-fitness criterion, and compare it to the concatenation of 35 short 16-second cyclic trajectories. As discussed earlier, our full dataset is 1800+4200=6000 trajectories. Thus we define a greedy strategy to pick a good subset 35 trajectories for direct comparison. To achieve this, we first compute the diagonal of $Y^T Y$ for each trajectory, which we shall denote by ψ_i for the ith trajectory.

Each trajectory is then scored the following metric d_i that balances preference for batch-diverse and exciting trajectories:

$$g_i = ||\psi_i - \psi_{\text{previous best}}|| \tag{5.10}$$

$$f_i = \text{sum}(\psi_i) \tag{5.11}$$

$$d_i = \frac{g_i}{\max(g)} + \frac{f_i}{\max(f)} \tag{5.12}$$

We greedily pick the trajectory with greatest score d_i, where 'previous best' is the previously selected best trajectory, and initially $\psi_{\text{previous best}}$ is set to zero. Thus we prefer those that are far from the previous choice, and have large diagonals. After each trajectory is selected, it gets removed from the set of trajectories to choose from next.

Setup: System Identification We next use recursive least squares (RLS) to perform system identification and learn the dynamics of a Kuka LWR IV platform using the conventional long, conventional short (35 of 1800), and extended SIDE-GAN (35 of 6000) generated trajectories. We use the fitted model to perform torque prediction and report the torque prediction accuracy (normalized mean squared error, nMSE) in Table 5.1. We repeat this experiment using both the simulated platform (perfect dynamic model with 10% uniform random noise) via ARDL library [174] and a real robotic arm.

Results The results show that, controlling for trajectory length: (1) Our multiple trajectory approaches clearly outperform the conventional single long trajectory approach both in simulation and on the real KUKA LWR. (2) The additional excitation trajectories synthesised by SIDE-GAN produce a small improvement over multiple trajectory optimisation.

In terms of compute requirements: The single 560s large trajectory generation required 14 hours, the 1800 seed trajectories ($\approx 8h$ length) required 30 hours to generate (parallelised), the SIDE-GAN required a further 40 minutes to train, but thereafter can generate short 16s trajectories in 1.4ms per trajectory per thread, compared to 6 minutes per short trajectory using the conventional optimisation.

Overall, we conclude that multi-trajectory optimisation performs favorably compared to the conventional approach, and especially with SIDE-GAN can easily be scaled to generating more excitation trajectory data for system identification. In the next section, we explore the benefit of using the full generated dataset for dynamics learning.

5.5.4 Exp 2: System Identification with SIDE-GAN

We next evaluate system identification performance when using our full SIDE-GAN generated dataset.

Trajectories:	Single Long	Multiple original	Multiple SIDE-GAN
Simulator	0.2248 (±0.056)	0.0063 (±0.002)	0.0028 ((± 0.001)
Real Robot	8.509 (±9.906)	0.0239 (±0.017)	0.0210 (±0.016)

Table 5.1 **Average nMSE across joints.** Multiple (35) short cyclic trajectories show better performance than the conventional single longer trajectory of comparable length. The best 35 short trajectories generated with SIDE-GAN further improve nMSE over those from the original seed set.

Torque Prediction We first evaluate torque prediction, as in the previous experiment. We compare the margin of improvement between: (1) Seed+GAN generated data, using several GAN variants and (2) The original seed data alone ($\times 1$), and (3) The seed data, replicated $\times 4$ or $\times 10$ times (each replication is done with 10% uniform random noise on both the robot state $(q, \dot{q}, \ddot{q})$ and on the output torques). The $\times 4$ replication corresponds to a similar amount of data to our GAN-generated dataset, and the $\times 10$ replication corresponds to significantly more data than our GAN-generated dataset.

The results in Table 5.2 are reported in terms of % improvement in torque prediction nMSE. We can see that: (1) Vanilla DCGAN already leads to a clear improvement, and (2) Our two SIDE-GAN variants further improve on vanilla DCGAN trajectory generation, (3) Comparing our two SIDE-GAN variants that use eigenvalue or diagonal-based fitness, the eigenvalue-fitness variant performs best. (4) Simply replicating the original seed data does provide a simple alternative: the margin of our methods over the original data results does not decrease systematically with replication factor.

Parameter Estimation Quality We next investigate the parameter estimation quality for the different methods. We quantify estimation quality by the norms of the diagonal of the covariance from the RLS algorithm, which gives an estimate of the uncertainty of the internal base parameters.

The results in Table 5.3 shows that SIDE-GAN eigenvalue-fitness variant provides the lowest (least uncertain) norm estimates compared to both the original data and any of the other competitors.

5.5.5 Discussion

SIDE-GAN Dependence on Seed Set Size Our experiments used a fixed seed set of 1800 trajectories throughout to train SIDE-GAN. We explored reducing the training set size. Training on, e.g., 240 trajectories, SIDE-GAN still generates diverse and exciting trajectories. However, this does not provide sufficient data for the GAN to learn the constraints well, and the validity rate of generated trajectories suffer (about 10%).

Original + generated data vs.	orig. ($\times$1)	orig. ($\times$4)	orig. ($\times$10)
+ Vanilla DCGAN	36% (±44%)	21% (±30%)	38% (±17%)
+ SIDE-GAN (Y^TY eigens)	**51%** (±27%)	**39%** (±25%)	**53%** (±8%)
+ SIDE-GAN (Y^TY diag.)	**52%** (±16%)	35% (±32%)	44% (±27%)

Table 5.2 **Improvements in nMSE of torque predictions** with respect to the original training data (fed into RLS $\times$1, $\times$4, and $\times$10 times with 10% uniform random noise). Improvement ratio is averaged across the joints.

Methods	Covariance Diagonals Norms
Original dataset $\times$1	2.65384
Original dataset $\times$4	2.44338
Vanilla DCGAN	2.2573
SIDE-GAN (Y^TY eigens)	**1.89143**
SIDE-GAN (Y^TY diag.)	2.36345

Table 5.3 **Parameter estimation quality.** Lower values mean that covariance matrices have smaller determinants, which means inertial parameters are predicted with more certainty.

This could still be useful since many samples can be drawn and invalid trajectories filtered. Generating and filtering in this way is still faster than conventional optimization (which takes roughly 6 minutes per trajectory) vs SIDE-GAN (32 per second, including checking the validity).

SIDE-GAN Generality GAN-based methods are generally indifferent to the specifics of the training data. Thus the SIDE-GAN method is expected to work well for other types of manipulators, and other dynamics models. That said, SIDE-GAN does need a seed set for the relevant manipulator. The trajectories generated by the SIDE-GANs are unlikely to generalise effectively across manipulators.

5.6 Conclusions and Future Work

This work shows the benefits of using multiple trajectories instead of the conventional single parameterised trajectory for the task of the system identification and torque prediction. Further, it proposes a method for generating valid and more diverse trajectories for the above task at the speed exceeding the underlying method by at least two orders of magnitude. The trajectory generator is trained to produce diversity in both

trajectory and inertial parameter space. Numerical results on trajectory validity, fitness metrics, and torque prediction – in both simulation and on real Kuka arm – confirm our contributions.

In future work we intend to investigate the use of conditional models to incorporate user-specified specific region-based and inertial-parameter-based focused exploration. We will also explore sparse data transfer learning to reduce the size of the seed set required to learn SIDE-GANs.

Chapter 6

Conclusion and Future Work

6.1 Contributions

The main contribution of this thesis is the investigation of using generative adversarial networks for previously overlooked applications in the field of robotics and automation. Specifically, rather than leveraging relatively well-explored image data generation, GANs can be of tremendous help in data augmentation for a variety of robotics and automation problems.

We have shown that GANs are very capable in addressing the data shortage. They synthesize diverse and realistic data for such settings as

(i) visually coherent sonar simulation;

(ii) data augmentation for underwater target recognition and detection systems

(iii) behavioural repertoires for robot control;

(iv) generation of specific kinds of trajectories for system identification.

The task has been two-fold for all of the applications presented - ensuring a reasonable quality and diversity of the generated samples, along with following the specific domain constraints of each individual setting. I.e. keeping the generation coherent and continuous for sonar data, making sure self-collisions and bridging the joint limits is minimal in the control applications. Both of these objectives were met for all of the presented scenarios.

In addition to this, a number of the novel methods were presented as a part of this thesis, most of these have already been published as conference papers and well-received by the Robotics community.

6.2 Assumptions and Limitations

GANs are after all a Deep Learning family of methods, and as such their training can be tricky.[1] However, the training should be smooth under the following assumptions concerning the training dataset:

(i) Sufficient training data is required to bootstrap the GAN.

(ii) The training data needs to be suitably representative of conditions of interest.

Most of the issues are caused by violating the above-mentioned assumptions, e.g.:

- *extremely small amounts of initial training data available, violation of the first assumption.* For all of the scenarios presented in this paper we have noticed that the proposed methods produce realistic looking results starting at around 200 of training examples. For the conditional models (e.g. for targeted throwing) this number might be higher depending on the variability of the problem-specific training condition.
  This is generally a fatal limitation - GANs are designed to learn to approximate data distributions and cannot learn them without sufficient initial training data present. In some specific cases it might be possible to bootstrap a GAN with some other method, such as QD search in Chapter 4.

- *very limited or no initial diversity* in the initial training data, a violation of the representativeness assumption. E.g. providing the same limited set of examples multiple times or even some variations of it with added noise is unlikely to resolve in a well-trained GAN.
  There is no way around lack of diversity - it always results in a *mode collapse*, hence one is bound to either go through with additional data collection to resolve this issue or to abandon the idea of using GANs. However, it is worth noting that pretty much any generative model would fail in this case.

- *highly unbalanced numbers of samples for certain target conditions, another violation of the representativeness assumption.* Very limited amount of examples for specific set of conditions could mean the trained model might not be able to generalise for such conditions.
  Occasionally, this might be possible to resolve by some form of data augmentation, depending on how well the data set generalises. For instance, we had some very limited examples of certain types of terrains in Chapter 3, and we managed to get

[1] More details have been provided in Chapter 2.

around this by artificially balancing the training set using augmented imagery, as well as cropping overlapping tiles for training (so some bits of the entire image would be used more than once with a variety of displacements).

Additionally, although a conventional problem of GANs usually is the lack of sensible evaluation techniques, is less applicable to our work. Since data augmentation for robotics and automation applications ultimately would be used in some further task, we have an advantage. We can always assess the quality of the synthesized data based on the performance improvement on a downstream task, when using this synthetic data.

However, a crucial limitation is that it is not always possible to differentiate a downstream task with respect to a GAN, in order to train the GAN to directly optimise on the end-task. We have been successful in partially surmounting this limitation in case of the SIDE-GANs by separately pre-training surrogate models (converter and predictor networks) and using them to influence the training of the rest of the architecture.

6.3 Future Work

There are some immediate extensions and alternative uses for some of the methods presented in this thesis:

- MC-pix2pix and R2D2-GAN (Chapter 3) can be used for the semantic segmentation of the underwater sonar data, using the same models and training data, by just reversing the direction of the image translation at training process. This might be useful for the downstream task of grading the complexity of terrain for the further use by ATR systems and human operators. Specifically, terrains with more complex patterns, such as various types of ripples, are more tricky for ATR to single out the objects and might require double-checks from human operators. We have provided some initial positive results in the end of the Chapter 3, however a much more detailed investigation would be beneficial for development of underwater ATR systems.

- A simplified versions of MC-pix2pix (Chapter 3) could be used for generating simple backgrounds in side-scrolling games in order to minimise the game designers efforts.

- R2D2-GANs (Chapter 3) can be instantaneously used for generating different types of maps - both aerial-view style and 3D game-style terrains. The only thing that would need to be replaced for map generation, compared to the current setting, is the training set. I.e., the R2D2 would need to be trained on the colour

and height maps either aerial-view or any of the preferred 3D video game terrain. The synthetic aerial views then can potentially be used to boost the performance of the aerial detection and tracking algorithms. The 3D game terrain GANs can be used as default world generators for massively multiplayer online role-playing games. Not only it would mean an easy starting point for game designers, but also potentially more memory efficiency. I.e., instead of storing the entire world map, one could store just a trained generator and a relatively small library of semantic maps and/or noise vectors corresponding to the patches in the game world, re-generating these patches as player accesses them.

- GPNs (Chapter 4) could serve as a reasonable bootstrapping technique for more sophisticated Reinforcement Learning algorithms, for instance by expanding the initial supervised pre-training datasets for the RL algorithms.

- GPNs (Chapter 4) can be extended to generate the closed loop policies, such as Dynamic Motion Primitives. These can be eventually used for the real-time adaptation to changing environments.

- SIDE-GANs (Chapter 5) can easily be extended to a spatially-restricted dynamic system ID setting, which is conditioned on a limited available action space. For example, running a dynamic system identification for a robot platform installed in a real industrial setting might come with space limitations, such as walls, furniture, and equipment in a close proximity to the robot. It is essential to explore the possibilities for efficient dynamic system ID in restricted spaces for a number of real-world applications.

6.4 General Implications of Using GANs

his thesis explores only a limited amount of some case-specific problems, but in reality the potential applications of GANs to resolve data shortages are numerous. In theory, nearly any downstream ML method struggling for larger number of training data could make use of a GAN-based architecture.

Beyond conceptual technical limitations listed in section 6.2 there are other practical implications that might affect GANs usefulness and usability. From the perspective of AI and Robotics there are the following criteria to be considered:

- *Suitability for the problem at hand* - GAN-based architecture might be a best solution if a variety of solutions is required or multiple attempts are allowed, like

in GPN or SIDE-GAN cases. It is however not necessarily the best solution for other scenarios. For instance, finding an optimal solution for a single task in a single environment calls for an RL method.

- More specifically, it often has to do with *trustworthiness* - GANs can be trained and fine-tuned to produce largely good quality solutions, however there is no guarantee from occasional outliers, nor an there be one. Hence, GANs are more useful as simulations for reference only (not the main source of information for decision making), supplementary to human expert in charge, or with some safety thresholds in place (e.g. checking the robot controllers in simulation before executing on a real robot to make sure the controller will not damage hardware).

- *Data gathering, costs, and availability* - assuming the above suitability consideration was taken into account, the data availability can represent a serious barrier. Things like manual labelling, or expense associated with the data gathering due to the cost (and in case of robot platforms often also fragility) of the equipment, amortisation costs, lack (or total absence) of sensory data from robots operating in wild (as opposed to the lab conditions), etc, can make result in data being expensive, commercially sensitive, private, or completely unavailable. The lack of a decent training set renders GANs useless, as discussed in section 6.2.

- *Addressing privacy and the lack of such* - GANs could be viewed as a great boost for medical applications of machine learning. As mentioned in the previous bullet point - patients private data (PII) are sensitive and often under the restricted access, and the limited availability of it often blocks the development of the ML models for medical data processing, diagnostics, etc.

  In theory, we could train a GAN and use it to generate realistic yet not sensitive training data for the downstream ML tasks without endangering patients private data. In practice there are no guarantees that a trained GAN did not memorize such data and will not spit them out at random when sampled. Moreover GANs have been proven to be at least somewhat susceptible to membership inference attacks (MIA) - identifying or even restoring the original training data samples from the trained model [175]. Which ultimately means that GANs in their current state are neither particularly secure nor private under the MIA attacks. (Please note: this issue is not related to the mode collapse, and only sometimes marginally related to overfitting. [176])

  Practically, it means that GANS are prone to the exact same data leakage issue they seemingly could solve, and that sharing a trained model can prove equivalent

to explicitly sharing the training data. There are case-based situations in which the sensitive training data can be sterilised prior to being used in GAN training, and that is how it is usually addressed currently. However a more global solution of this issue is still a work in progress.

That said, and although there is still a great deal of improvements to make, GANs are one of the most powerful generative models in Machine Learning and are already proving useful in a number of domains.

References

[1] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *ArXiv*, 2014.

[2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[3] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[5] Marija Jegorova, Stéphane Doncieux, and Timothy M. Hospedales. Generative Adversarial Policy Networks for Behavioural Repertoire. In *International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, 2019.

[6] Marija Jegorova, Antti Ilari Karjalainen, Jose Vazquez, and Timothy Hospedales. Full-Scale Continuous Synthetic Sonar Data Generation with Markov-Conditional Generative Adversarial Networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[7] Marija Jegorova, Antti Ilari Karjalainen, Jose Vazquez, and Timothy Hospedales. Unlimited Resolution Image Generation with R2D2-GANs. *IEEE OCEANS*, 2020.

[8] Marija Jegorova, Joshua Smith, Michael Mistry, and Timothy Hospedales. Adversarial Generation of Informative Trajectories for Dynamics System Identification. *ArXiv*, 2020.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[10] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray

Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, pages 484–489, January 2016.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, June 2019.

[12] Gary Marcus. Deep Learning: A Critical Appraisal. *CoRR*, 2018.

[13] Tjeerd van der Ploeg, Peter Austin, and Ewout Steyerberg. Modern Modelling Techniques are Data Hungry: A Simulation Study for Predicting Dichotomous Endpoints. *BMC Medical Research Methodology*, 2014.

[14] Han Altae-Tran, Bharath Ramsundar, Aneesh Pappu, and Vijay Pande. Low Data Drug Discovery with One-Shot Learning. *ACS Central Science*, 3, 2016.

[15] Jiaxing Tan. *Deep Learning Based Medical Image Analysis with Limited Data*. PhD thesis, City University of New York (CUNY), 2019.

[16] Ken Wong, Tanveer Syeda-Mahmood, and Mehdi Moradi. Building Medical Image Classifiers with Very Limited Data Using Segmentation Networks. *Medical Image Analysis*, 49, 2018.

[17] Harry Pierson and Michael Gashler. Deep Learning in Robotics: A Review of Recent Research. *Advanced Robotics*, 2017.

[18] Babak Akhgar, Gregory B. Saathoff, Hamid R. Arabnia, Richard Hill, Andrew Staniforth, and Petra Saskia Bayerl. *Application of Big Data for National Security: A Practitioner's Guide to Emerging Technologies*. Butterworth-Heinemann, USA, 1st edition, 2015.

[19] Michael Veale and Reuben Binns. Fairer machine learning in the real world: Mitigating discrimination without collecting sensitive data. *Big Data & Society*, 2017.

[20] Milad Nasr, Reza Shokri, and Amir Houmansadr. Machine Learning with Membership Privacy Using Adversarial Regularization. In *ACM SIGSAC Conference on Computer and Communications Security*, page 634–646. Association for Computing Machinery, 2018.

[21] Yuji Roh, Geon Heo, and Steven Whang. A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[22] Sergey Levine, Peter Pastor Sampedro, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. In *International Symposium on Experimental Robotics*, 2017.

[23] Ashesh Jain, Avi Singh, Hema Koppula, Shane Soh, and Ashutosh Saxena. Recurrent Neural Networks for Driver Activity Anticipation via Sensory-Fusion Architecture. *ArXiv*, 2015.

[24] Lerrel Pinto and Abhinav Gupta. Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours. *CoRR*, 2015.

[25] Tatiana López Guevara, Rita Pucci, Nicholas Kenelm Taylor, Michael Gutmann, Subramanian Ramamoorthy, and Kartic Subr. To Stir or Not to Stir: Online Estimation of Liquid Properties for Pouring Actions. *Workshop on Learning and Inference in Robotics: Integrating Structure, Priors and Models*, 2018.

[26] S. Calinon, F. Guenter, and A. Billard. On Learning, Representing and Generalizing a Task in a Humanoid Robot. *IEEE Transactions on Systems, Man and Cybernetics, Special issue on robot learning by observation, demonstration and imitation*, 2006.

[27] Nadia Figueroa and Aude Billard. A Physically-Consistent Bayesian Non-Parametric Mixture Model for Dynamical System Learning. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Conference on Robot Learning (CORL)*, pages 927–946. PMLR, 2018.

[28] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. Reset-Free Trial-and-Error Learning for Robot Damage Recovery. *Robotics and Autonomous Systems*, pages 236 – 250, 2018.

[29] Sylvain Koos, Antoine Cully, and Jean-Baptiste Mouret. Fast Damage Recovery in Robotics with the T-Resilience Algorithm. *The International Journal of Robotics Research*, 2013.

[30] S.J. Pan and Q. Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1345–1359, 2010.

[31] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning. *ArXiv*, 2019.

[32] Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. Few-Shot Learning Through an Information Retrieval Lens. pages 2255–2265, 2017.

[33] Massimiliano Patacchiola, Jack Turner, Elliot Crowley, and Amos Storkey. Deep Kernel Transfer in Gaussian Processes for Few-shot Learning. *ArXiv*, 2019.

[34] Yan Wang, Wei-Lun Chao, Kilian Weinberger, and Laurens van der Maaten. SimpleShot: Revisiting Nearest-Neighbor Classification for Few-Shot Learning. *ArXiv*, 2019.

[35] Xin Hu, Jingjing Chen, Fei Wang, and Dan Zhang. Ten challenges for EEG-based affective computing. *Brain Science Advances*, 2019.

[36] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Are We Making Real Progress in Simulated Environments? Measuring the Sim2Real Gap in Embodied Visual Navigation. *ArXiv*, 2019.

[37] Jean-Baptiste Mouret and Konstantinos Chatzilygeroudis. 20 Years of Reality Gap: A Few Thoughts about Simulators in Evolutionary Robotics. In *Genetic and Evolutionary Computation Conference (GECCO)*, page 1121–1124. Association for Computing Machinery, 2017.

[38] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stanley T. Birchfield. Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1082–10828, 2018.

[39] OpenAI Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. *The International Journal of Robotics Research*, 2019.

[40] Connor Shorten and Taghi Khoshgoftaar. A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6, 2019.

[41] N. Davis and K. Suresh. Environmental Sound Classification Using Deep Convolutional Neural Networks and Data Augmentation. In *IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pages 41–45, 2018.

[42] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. AutoAugment: Learning Augmentation Policies from Data.

[43] Kaiyang Zhou, Yongxin Yang, Timothy Hospedales, and Tao Xiang. Deep Domain-Adversarial Image Generation for Domain Generalisation. In *Conference on Artificial Intelligence (AAAI)*. American Association for Artificial Intelligence (AAAI), 2019.

[44] Rok Blagus and Lara Lusa. SMOTE for High-Dimensional Class-Imbalanced Data. In *BMC Bioinformatics*, 2012.

[45] Haibo He, Yang Bai, Edwardo Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. pages 1322 – 1328, 2008.

[46] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond Empirical Risk Minimization. In *International Conference on Learning Representations (ICLR)*, 2018.

[47] Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *Annals of Mathematical Statistics*, 27, 1956.

[48] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *International Conference on Learning Representations (ICLR)*, 2014.

[49] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. SinGAN: Learning a Generative Model From a Single Natural Image. *International Conference on Computer Vision (ICCV)*, pages 4569–4579, 2019.

[50] Kyungjune Baek, Duhyeon Bang, and Hyunjung Shim. Editable Generative Adversarial Networks: Generating and Editing Faces Simultaneously. *Asian Conference on Computer Vision (ACCV)*, pages 39–55, 2019.

[51] Wonkwang Lee, Donggyun Kim, Seunghoon Hong, and Honglak Lee. High-Fidelity Synthesis with Disentangled Representation. *ArXiv*, 2020.

[52] T Jaydeep. Comparative Study of GAN and VAE. *International Journal of Computer Applications*, 182, 2018.

[53] Antreas Antoniou, Amos J. Storkey, and Harrison A Edwards. Data Augmentation Generative Adversarial Networks. *ArXiv*, 2017.

[54] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Augmenting Image Classifiers Using Data Augmentation Generative Adversarial Networks. *International Conference on Artificial Neural Networks (ICANN)*, pages 594–603, 2018.

[55] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 105–114, 2017.

[56] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2016.

[57] Jiahui Yu, Zhe L. Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Generative Image Inpainting with Contextual Attention. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[58] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[59] Haoye Cai, Chunyan Bai, Yu-Wing Tai, and Chi-Keung Tang. Deep Video Generation, Prediction and Completion of Human Action Sequences. In *The European Conference on Computer Vision (ECCV)*, 2018.

[60] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *International Conference on Computer Vision (ICCV)*, pages 2242–2251, 2017.

[61] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic Image Synthesis With Spatially-Adaptive Normalization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[62] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[63] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems*. 2016.

[64] Yunzhu Li, Jiaming Song, and Stefano Ermon. Inferring The Latent Structure of Human Decision-Making from Raw Visual Inputs. *ArXiv*, 2017.

[65] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved Techniques for Training GANs. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

[66] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative Adversarial Text to Image Synthesis. In *International Conference on Machine Learning (ICML)*, page 1060–1069. JMLR.org, 2016.

[67] Jost Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. 2015.

[68] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[69] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks, 2015.

[70] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines. volume 27, pages 807–814, 06 2010.

[71] L Arjovsky, M.; Chintala S.; Bottou. Wasserstein GAN. *International Conference on Machine Learning (ICML)*, 2017.

[72] L. Theis, A. van den Oord, and M. Bethge. A Note on the Evaluation of Generative Models. In *International Conference on Learning Representations (ICLR)*, 2016.

[73] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[74] Michael Gutmann and Aapo Hyvärinen. Noise-Contrastive Estimation: A New Estimation Principle for Unnormalized Statistical Models. In *International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, pages 297–304. PMLR, 2010.

[75] Jascha Sohl-Dickstein, Peter Battaglino, and Michael R. DeWeese. Minimum Probability Flow Learning. In *International Conference on Machine Learning (ICML)*, 2011.

[76] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J. Smola. A Kernel Method for the Two-Sample-Problem. In *Advances in Neural Information Processing Systems 19*, pages 513–520. MIT Press, 2007.

[77] Yujia Li, Kevin Swersky, and Richard Zemel. Generative Moment Matching Networks. In *International Conference on Machine Learning (ICML)*, page 1718–1727. JMLR.org, 2015.

[78] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition,*, 2016.

[79] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[80] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved Training of Wasserstein GANs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., 2017.

[81] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. F-GAN: Training Generative Neural Samplers Using Variational Divergence Minimization. In *International Conference on Neural Information Processing Systems (NeurIPS)*, page 271–279. Curran Associates Inc., 2016.

[82] Mario Lučić, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs Created Equal? A Large-Scale Study. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[83] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *International Conference on Learning Representations (ICLR)*, 2019.

[84] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image Inpainting for Irregular Holes Using Partial Convolutions. *ECCV*, 2018.

[85] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *ArXiv*, 2018.

[86] Md. Akmal Haidar and Mehdi Rezagholizadeh. TextKD-GAN: Text Generation Using Knowledge Distillation and Generative Adversarial Networks. *32nd Canadian Conference on Artificial Intelligence*, 2019.

[87] Sandeep Subramanian, Sai Rajeswar Mudumba, Alessandro Sordoni, Adam Trischler, Aaron C Courville, and Chris Pal. Towards Text Generation with Adversarially Learned Neural Outlines. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7551–7563. Curran Associates, Inc., 2018.

[88] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[89] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-Video Synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[90] Wan, Chia-Hung and Chuang, Shun-Po and Lee, Hung-Yi. Towards audio to scene image synthesis using generative adversarial network. *ArXiv*, 2018.

[91] Konstantinos Vougioukas, Stavros Petridis, and Maja Pantic. End-to-End Speech-Driven Facial Animation with Temporal GANs. In *The British Machine Vision Conference (BMVC)*, 2018.

[92] Tony Jebara. *Machine Learning: Discriminative and Generative*. Kluwer Academic Publishers, USA, 2003.

[93] Andrew Y. Ng and Michael I. Jordan. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press, 2002.

[94] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, Inc., USA, 4th edition, 2008.

[95] Judea Pearl. Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning. 1985.

[96] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[97] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Willey & Sons, New Yotk, 1973.

[98] *Pearson, Karl*, pages 418–419. Springer New York, New York, NY, 2008.

[99] Carlos Amendola, Jean-Charles Faugere, and Bernd Sturmfels. Moment Varieties of Gaussian Mixtures. *Journal of Algebraic Statistics*, 7(1), 2016.

[100] L. Yao and Z. Ge. Scalable Semisupervised GMM for Big Data Quality Prediction in Multimode Processes. *IEEE Transactions on Industrial Electronics*, 66(5):3681–3692, 2019.

[101] Raina Robeva, Aaron Garrett, James Kirkwood, and Robin Davies. Chapter 9: Identifying CpG Islands: Sliding Window and Hidden Markov Model Approaches. In *Mathematical Concepts and Methods in Modern Biology*, pages 267 – 305. Academic Press, Boston, 2013.

[102] Chapter 9: Entity Extraction. In Gary Miner, Dursun Delen, John Elder, Andrew Fast, Thomas Hill, and Robert A. Nisbet, editors, *Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications*, pages 921 – 928. Academic Press, Boston, 2012.

[103] Byung-Jun Yoon. Hidden Markov Models and their Applications in Biological Sequence Analysis. *Current Genomics*, 10:402–15, 2009.

[104] Ryouhei Kawamoto, Alwis Nazir, Atsuyuki Kameyama, Takashi Ichinomiya, Keiko Yamamoto, Satoshi Tamura, Mayumi Yamamoto, Satoru Hayamizu, and Yasutomi Kinosada. Hidden Markov Model for Analyzing Time-Series Health Checkup Data. *Studies in health technology and informatics*, 192:491–5, 2013.

[105] Mengqi Zhang, Xin Jiang, Zehua Fang, Yue Zeng, and Ke Xu. High-Order Hidden Markov Model for Trend Prediction in Financial Time Series. *Physica A: Statistical Mechanics and its Applications*, 517, 2018.

[106] Nicolas Städler and Sach Mukherjee. Penalized Estimation in High-Dimensional Hidden Markov Models with State-Specific Graphical Models. *Annals of Applied Statistics*, 7(4):2157–2179, 2013.

[107] Geoffrey I Webb, Janice R Boughton, and Zhihai Wang. Not so naive Bayes: aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, 2005.

[108] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research (JMLR)*, 3:993–1022, 2003.

[109] Zhipeng Wang and David Scott. Nonparametric Density Estimation for High-Dimensional Data - Algorithms and Applications. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2019.

[110] Geoffrey Hinton. Deep Belief Networks. *Scholarpedia*, 4, 2009.

[111] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. page 77, 2009.

[112] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1096–1104. Curran Associates, Inc., 2009.

[113] Yuming Huang, Ashkan Panahi, Hamid Krim, Yiyi Yu, and Spencer Smith. Deep Adversarial Belief Networks. *ArXiv*, 2019.

[114] Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann Machines. In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455. PMLR, 2009.

[115] Geoffrey E Hinton and Russ R Salakhutdinov. A Better Way to Pretrain Deep Boltzmann Machines. In *Advances in Neural Information Processing Systems 25*, pages 2447–2455. Curran Associates, Inc., 2012.

[116] Jungang Xu, Hui Li, and Shilong Zhou. An Overview of Deep Generative Models. *IETE Technical Review*, 32:131–139, 2014.

[117] Bin Dai and David Wipf. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations*, 2019.

[118] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation. *CoRR*, 2014.

[119] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. 2017.

[120] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In *NeurIPS*, 2018.

[121] Augustus Odena. Open Questions about Generative Adversarial Networks. *Distill*, 2019. https://distill.pub/2019/gan-open-problems.

[122] Cristóbal Esteban, Stephanie Hyland, and Gunnar Rätsch. Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs. *ArXiv*, 2017.

[123] Antti Ilari Karjalainen, Roshenac Mitchell, and Jose Vazquez. Training and Validation of Automatic Target Recognition Systems using Generative Adversarial Networks. *Sensor Signal Processing for Defence*, 2019.

[124] Cameron Fabbri, Md Jahidul Islam, and Junaed Sattar. Enhancing Underwater Imagery Using Generative Adversarial Networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7159–7165, 2018.

[125] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:318–327, 2020.

[126] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[127] Onur Tasar, S L Happy, Yuliya Tarabalka, and Pierre Alliez. ColorMapGAN: Unsupervised Domain Adaptation for Semantic Segmentation Using Color Mapping Generative Adversarial Networks. *IEEE Transactions on Geoscience and Remote Sensing*, 2020.

[128] Yuan Xue, Tao Xu, Han Zhang, Rodney Long, and Xiaolei Huang. SegAN: Adversarial Network with Multi-scale L_1 Loss for Medical Image Segmentation. *Neuroinformatics*, 16, 2017.

[129] G Fagogenis, V De Carolis, and D M Lane. Online fault detection and model adaptation for Underwater Vehicles in the case of thruster failures. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[130] Kuan Fang, Yunfei Bai, Stefan Hinterstoisser, Silvio Savarese, and Mrinal Kalakrishnan. Multi-Task Domain Adaptation for Deep Learning of Instance Grasping from Simulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[131] Chenyang Zhao, Timothy M Hospedales, Freek Stulp, and Olivier Sigaud. Knowledge Transfer Across Skill Categories for Robot Control. *IJCAI*, 2017.

[132] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean Baptiste Mouret. Robots That Can Adapt Like Animals. *Nature*, 521:503–507, 2015.

[133] A Cully and J B Mouret. Evolving a behavioral repertoire for a walking robot. *Evolutionary Computation*, 24:59–88, 2016.

[134] M Duarte, J Gomes, S M Oliveira, and A L Christensen. Evolution of Repertoire-Based Control for Robots With Complex Locomotor Systems. *IEEE Transactions on Evolutionary Computation*, 22:314–328, 2018.

[135] F. Guerin, N. Kruger, and D. Kraft. A Survey of the Ontogeny of Tool Use: From Sensorimotor Experience to Planning. *IEEE Transactions on Autonomous Mental Development*, 5(1):18–45, 2013.

[136] E. Ugur, Y. Nagai, E. Sahin, and E. Oztop. Staged Development of Robot Skills: Behavior Formation, Affordance Learning and Imitation with Motionese. *IEEE Transactions on Autonomous Mental Development*, 7(2):119–139, 2015.

[137] D. Kraft, R. Detry, N. Pugeault, E. Baseski, F. Guerin, J. H. Piater, and N. Kruger. Development of Object and Grasping Knowledge by Robot Exploration. *IEEE Transactions on Autonomous Mental Development*, 2(4):368–383, 2010.

[138] Joel Lehman and Kenneth O Stanley. Evolving a Diversity of Virtual Creatures Through Novelty Search and Local Competition. *Genetic and Evolutionary Computation Conference (GECCO)*, 2011.

[139] Andy Clark and Annette Karmiloff-Smith. The Cognizer's Innards: A Psychological and Philosophical Perspective on the Development of Thought. *Mind & Language*, 8(4):487–519, 1993.

[140] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *International Conference on Neural Information Processing Systems (NeurIPS)*, pages 2672–2680, 2014.

[141] Andras Gabor Kupcsik, Marc Peter Deisenroth, Jan Peters, and Gerhard Neumann. Data-efficient Generalization of Robot Skills with Contextual Policy Search. In *Conference on Artificial Intelligence (AAAI)*, 2013.

[142] J Kober, E Oztop, and J Peters. Reinforcement Learning to adjust Robot Movements to New Situations. In *RSS*, 2010.

[143] F Stulp, G Raiola, A Hoarau, S Ivaldi, and O Sigaud. Learning compact parameterized skills with a single regression. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013.

[144] Ronald J Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 1992.

[145] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2016.

[146] Tatiana Lopez-Guevara, Nicholas K Taylor, Michael U Gutmann, Subramanian Ramamoorthy, and Kartic Subr. Adaptable Pouring: Teaching Robots Not to Spill using Fast but Approximate Fluid Simulation. In *Conference on Robot Learning (CORL)*, 2017.

[147] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. In *arXiv:1712.06567*, 2018.

[148] F. Stulp, E. Oztop, P. Pastor, M. Beetz, and S. Schaal. Compact Models of Motor Primitive Variations for Predictable Reaching and Obstacle Avoidance. In *IEEE-RAS International Conference on Humanoid Robots*, 2009.

[149] Joel Lehman and Kenneth O Stanley. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation*, 19:189–222, 2011.

[150] A Cully and Y Demiris. Quality and Diversity Optimization: A Unifying Modular Framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2018.

[151] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[152] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills Without a Reward Function. *International Conference on Learning Representations (ICLR)*, 2019.

[153] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *International Conference on Learning Representations (ICLR)*, 2016.

[154] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *ArXiv*, 2014.

[155] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative Adversarial Text to Image Synthesis. *International Conference on Machine Learning (ICML)*, 2016.

[156] Xi Chen, Yan Duan, Rein Houthooft nad John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *International Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[157] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. *International Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[158] Zackory Erickson, Sonia Chernova, and Charles C Kemp. Semi-Supervised Haptic Material Recognition for Robots using Generative Adversarial Networks. *Conference on Robot Learning (CORL)*, 2017.

[159] Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-To-End Learning from Demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[160] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In *International Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[161] A. Cully and Y. Demiris. Quality and Diversity Optimization: A Unifying Modular Framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2018.

[162] Seungsu Kim, Alexandre Coninx, and Stephane Doncieux. From Exploration to Control: Learning Object Manipulation Skills Through Novelty Search and Local Adaptation. *ArXiv*, 2019.

[163] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[164] N. Jaquier. Improving the Drawing Skills of a Humanoid Robot with Visual Feedback, 2016.

[165] Jan Swevers, Chris Ganseman, D Bilgin Tukel, Joris De Schutter, and Hendrik Van Brussel. Optimal Robot Excitation and Identification. *IEEE transactions on robotics and automation*, pages 730–740, 1997.

[166] Kyung-Jo Park. Fourier-Based Optimal Excitation Trajectories for the Dynamic Identification of Robots. *Robotica*, pages 625–633, 2006.

[167] Stefan Bethge, Jörn Malzahn, Nikolaos Tsagarakis, and Darwin Caldwell. FloBaRoID—A software package for the identification of robot dynamics parameters. In *International Conference on Robotics in Alpe-Adria Danube Region*, pages 156–165. Springer, 2017.

[168] Hailin Ren and Pinhas Ben-Tzvi. Learning Inverse Kinematics and Dynamics of a Robotic Manipulator Using Generative Adversarial Networks. *Robotics and Autonomous Systems*, 2019.

[169] Chirstopher G. Atkeson, Chae H An, and John M Hollerbach. Estimation of Inertial Parameters of Manipulator Loads and Links. *The International Journal of Robotics Research*, 1986.

[170] M. Gautier. Numerical calculation of the base inertial parameters of robots. *Journal of Robotic Systems*, pages 485–506, 1991.

[171] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Bradford Company, USA, 2004.

[172] Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, pages 385–482, 2003.

[173] Francesco Biscani and Dario Izzo. esa/pagmo2: pagmo 2.13.0, January 2020.

[174] Joshua Smith and Michael Mistry. Online Simultaneous Semi-Parametric Dynamics Model Learning. *IEEE Robotics and Automation Letters*, 2020.

[175] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Evaluating privacy leakage of generative models using generative adversarial networks. 05 2017.

[176] Vaishnavh Nagarajan, Colin Raffel, and Ian J. Goodfellow. Theoretical insights into memorization in gans. *Neural Information Processing Systems Workshop*, 2018.

Appendix A

Additional Figures for Chapter 4

Figure 4.5 with respect to KDE (QD is provided for reference)

The following figure A.1 is an extension of Figure 4.6, where GPN is compared not only against the QD success rates, but also against the KDE.

Figure A.1 shows heat-maps of $SuccessesProportion(k, \tau = 0.2)$ for various occlusion rates and minimum hit requirements k. From these plots we can make the following observations:

(i) All methods have higher success rate in the easier bottom left (low occlusion, low hit ratio required for success), and vice-versa in the harder top right.

(ii) The GPN result is much higher than that of QD for low k values (e.g., $k = 1$), but also lower than KDE for $k = 1$, and almost equivalent to KDE for $k = 1$.

(iii) At very high minimum hit (e.g., $k = 9$) QD performance is slightly better than GPN. This is because GPNs slightly lower accuracy means that it's rarely the case that as many as 9 out of 10 attempts hit the target. However, at this stringent hit rate requirement, we note that the success rate of QD in absolute terms is also very low (around 10%).

(iv) The differences between GPN and KDE and GPN and QD (right column) show that the GPN can often find at least one way to hit the target, for this whole range of occlusion rates, but it is ultimately better than both competitors at the medium range of minimum required hits, i.e. $k = 3, 4, ..., 7$.

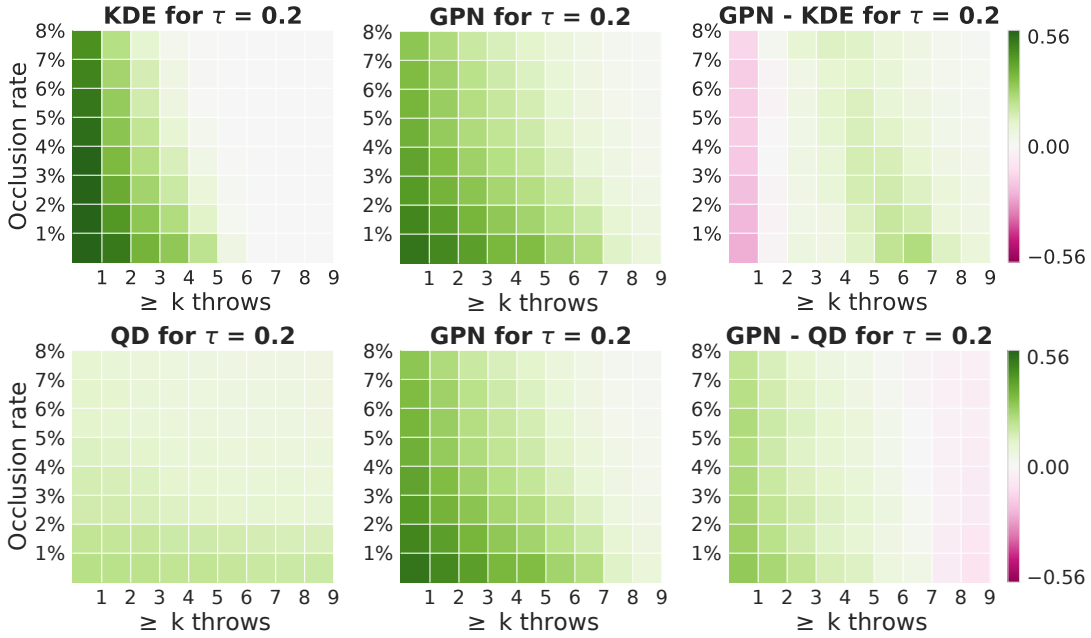

Fig. A.1 Robustness of target-conditional throwing in obstacle-occluded environments. Heat maps illustrate the probability of at least k of 10 throws landing within $\tau = 0.2$ of the target for different levels of occlusion. Top: KDE method. Our GPN method. Difference between them. Bottom: QD lookup method. Our GPN method. Difference between the GPN and QD results. Overall: it looks like KDE is the best for at least one successful hit out of 10, but worse than GPN for anything higher than that. QD might be the best solution if only very large amount of successful hits, e.g. 8 or 9 out of 10 are considered a success. GPN takes the middle ground from minimum required 2 to 7 successful hits out of 10.